

لغة السى من الأساسيات إلى الإحتراف

تأليف محمود البربرى



سى ببساطه

SIMPLY C

" قُلْ إِنَّ صَلاَتِي وَنُسُكِي وَمَحْيَايَ وَمَمَاتِي لِلّهِ رَبِّ الْعَالَمِينَ لاَ شَرِيكَ لَهُ وَبِذَلِكَ أُمِرْتُ وَأَنَاْ أَوَّلُ الْمُسْلِمِينَ "

إهداء

إلى والديّ رحمة الله عليهما ..

و إلى إخوتي الذين لا يدخرون جهداً في مساعدتي ، أحمل لهم العرفان دوماً ..

و إلى كل من حثني يوماً على النجاح ..

و إلى كل من يساهم في النهوض العلمي و المعرفي للأمة الإسلامية ..

نبذة عن الكتاب

الكتاب يحتوى على أساسيات لغة السى ، و يتطرق إلى بعض المواضيع المتقدمة المنتقاة ، و لا يتطرق إلى كل المواضيع ، لأنه تم عمل الكتاب ليشكل مصدراً لتعلم لغة السى بكل بساطة و وضوح فى الشرح لكل الأساسيات ، وليضعك على بداية طريق الإحتراف من خلال التطرق للموضوعات المتقدمة التى أدرجت فيه . و لم يتم عمله ليصبح مرجعاً يحتوى على كل المواضيع.

الكتاب يركز بشكل ملحوظ على المواضيع التى تختلف فيها لغة السى عن أغلب اللغات ، فستجد أن الفصول الأربعة الأخيرة (المؤشرات و حجز الذاكرة ديناميكياً ، الدوال ، المتغيرات النصية و التعامل مع الملفات) تم تركيز الشرح عليهم بشكل ملحوظ ، و الفصول الأولى تم تناولها بأبسط شكل ممكن لسببين: الأول/ الأغلبية عندهم علم مسبق بتلك المواضيع فهى مشابهة كثيراً لإستخدامها فى اللغات الأخرى . ثانياً/ سيتم توظيف هذه المواضيع فى برامجنا فى المواضيع المتقدمة ، لذلك لم أجد فائدة كبيرة فى إدراج أمثلة كثيرة فى المواضيع الأولى.

لتحقق الإستفادة القصوى من الكتاب يجب أن تكون قد تعاملت مع لغة برمجة واحدة على الأقل من قبل ، و لا يستلزم أى معرفة مسبقة بلغة السى .

لما قُمت بتأليف هذا الكتاب ؟

أنا طالب فى الفرقة الثالثة من كلية هندسة المنصورة قسم حاسبات و نظم التحكم ، مهتم بمجال الأنظمة المدمجة (Embedded Systems) ، و بدأت منذ عام تقريباً خطواتى الأولى فى هذا المجال سواء عملياً عن طريق المشاركة فى الفرق العملية بالجامعة ، أو علمياً عن طريق البدء فى دراسة لغة السى من المراجع المختصة ، و كنت قد انتهيت عند كتابة هذا الكتاب من دراسة مرجعين من وجهة نظرى الخاصة من أفضل المراجع التى كُتبت فى اللغة و تم ترشيحهم فى نهاية الكتاب لمن يريد الإطلاع عليهم ، و أطلعت على المؤلفات التى كتبت فى لغة السى باللغة العربية ، و وجدتها تفتقر إلى نموذج مشابه لهذا الكتاب ، فقررت البدء فى كتابة هذا الكتاب – أول كتاب شخصى لىّ ، و أودّ أن ينتفع به و لو شخص واحد فقط ، و أن يجعله الله خالصاً لوجهه الكريم .

كيفية عرض فصول الكتاب

1) يبدأ كل فصل بعرض " ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل " .

2) ثم يتم تناول نظرة عامة عن الخاصية التى سيتم دراستها فى هذا الفصل ، و إيضاح مجموعة من المفاهيم التى تساعدك فى فهم هذه الخاصية و الهدف من وراء دراستها .. إلخ .

3) ثم يتم شرح هذه الخاصية و التوضيح بمجموعة من الأمثلة المتنوعة .

4) ثم يتم تناول شرح برنامج تطبيقى – فى أغلب المواضيع - نقوم فيه بتعلم كيفية توظيف الخاصية التى تم دراستها فى هذا الفصل فى برامجنا الخاصة.

الفهرس

الفصل الأول

عن اللغة و البيئة التطويرية

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ مميزات و عيوب اللغة .
- ✓ المجالات التطبيقية للغة.
- ✓ التعرف على البيئة التطويرية Code::Blocks .
- ✓ عمل أول برنامج لك بإستخدام لغة السى.

مميزات اللغة

تعتبر لغة السى من أقوى اللغات على الإطلاق، و طُورت عنها العديد من اللغات الحديثة نسبياً مثل C# و جافا و C++، و هى لغة high-level ، ولكنها تحتوى على بعض خصائص الـ low-level ، لذا يطلق عليها فى الغالب middle-level language ، و سنستعرض معاً بعض مميزات اللغة التى جعلتها تحظى بذلك الرواج و بتلك القوة :

الكفاءة ، و هذا يرجع إلى أن اللغة low-level مقارنة باللغات الأخرى ، لما تحتويه من بعض الخصائص التى تتعامل مباشرة مع الهاردوير مثل المؤشرات - pointers ، مما يعنى أنها قريبة جداً من لغة الآلة ، و هذا بدوره يعنى أنها تقوم بتنفيذ البرامج بشكل أسرع ، و سرعة تنفيذ البرامج تُعد عاملاً مهماً فى تحديد قوة اللغة.

القوة ، و على الرغم من صغر لغة السى إلا أنها تستمد قوتها من الـ standard library الخاصة بها، و التى تحوى مئات الدوال التى تقوم بعمليات كثيرة، فتغنيك عن كتابة المئات من الأسطر للقيام بعملية معينة ، فهى تمدك بدالة تقوم بتلك العملية فى سطر واحد .

Portability, و تعنى أن البرنامج الذى تمت كتابته باللغة يعمل على مختلف أجهزة الحاسب الآلى بداية من الحاسب الشخصى و إنتهاءاً بالحاسبات العملاقة.

المرونة ، تتميز اللغة بأنها لا تحد المبرمج بحدود صارمة عندما يتعلق الأمر بإستخدام خواص اللغة و هذا يميزها عن العديد من اللغات، فلغة السى بنيت على قاعدة مشهورة تقول بأن ((المبرمج يعلم ما يفعل)) ، فهناك بعض العمليات تسمح بها اللغة و لا تعتبرها كخطأ و لكن فى لغات برمجية أخرى لا يتم السماح بها ، فمثلاً يمكنك جمع متغير من النوع char على آخر من النوع int أو float دون أى مشكلة ، هناك لغات أخرى لا تسمح بذلك ، و الأمثلة على تلك العمليات كثيرة و سيتضح لك الكثير منها خلال تعاملك مع اللغة ، و لكن هذه المرونة قد تتسبب لك فى بعض المشاكل فى برامجك – bugs.

التكامل مع نظام التشغيل Linux ، و هذا التكامل أضاف إلى اللغة الكثير و خاصة فيما يتعلق بالـ Portability.

عيوب اللغة

و كأى لغة برمجة، لا تخلو السى كذلك من العيوب، نستعرض بعضها معاً:

عرضة اكثر للأخطاء البرمجية، و هذا يرجع إلى أن المترجم الخاص باللغة لا يكتشف بعض الأخطاء أثناء عملية الترجمة و التى قد يتم إكتشف مثلها فى لغات أخرى، فمعظم الأخطاء تظهر فى وقت تنفيذ البرنامج و ليس ترجمته. و هذا هو الأثر السلبي للمرونة التى تتميز بها اللغة.

صعوبة فهم الكود، فعلى الرغم من صغر لغة السى إلا أنها تحتوى على خصائص عديدة تكتسبها من الـ standard library الخاصة بها كما ذكرنا من قبل ، و عند مزج هذه الخصائص معاً فى المراحل المتقدمة من إحتراف اللغة، تصبح الامور أكثر صعوبة.

صعوبة تعديل الكود، فى المشاريع الضخمة المطورة بلغة السى يصعب عليك تعديلها مرة أخرى لأن السى لا تدعم بعض الخصائص التى تساعد على تجزئة البرنامج و تنظيمه مثل الـ classes و الـ packages.

أهم المجالات التطبيقية للغة

تطوير نظم التشغيل – operating systems ، و تعد أهم المجالات التطبيقية للغة ، فمعظم نظم التشغيل التى نستخدمها يومياً حتى على هواتفنا المحمولة مطور جزء كبير منها بلغة السى.

الأنظمة المدمجة – embedded systems ، يتم برمجة الأنظمة المدمجة بلغة السى فى أغلب الأحيان ، و هى اللغة الأشهر فى هذا المجال ، فبرمجة المتحكمات الدقيقية – microcontrollers تتم بلغة السى أو بلغة مُكون معظمها من لغة السى، فمثلا الأردوينو يستخدم لغة خاصة به تسمى Arduino C و هى لغة مطورة من السى بشكل أساسى مع لغة أخرى تسمى الـ Processing .

تطوير compilers للغات أخرى ، و المترجم – compiler هو برنامج يترجم الكود إلى لغة الآلة .

تستخدم فى قواعد البيانات – databases ، و كذلك تستخدم فى تطوير الـ text editors مثل الـ word .

و هنا نكتفى بهذا القدر من التحدث عن مميزات اللغة و تطبيقاتها، و أعتقد بأنك تستطيع الآن أن تحدد حاجتك إلى تعلم اللغة من عدمها، و ننتقل الآن إلى الجزء الثانى من هذا الفصل الذى سنتعرف فيه على البيئة التطويرية التى سنستخدمها و سنقوم بكتابة أول برنامج بلغة السى.

البيئة التطويرية- IDE

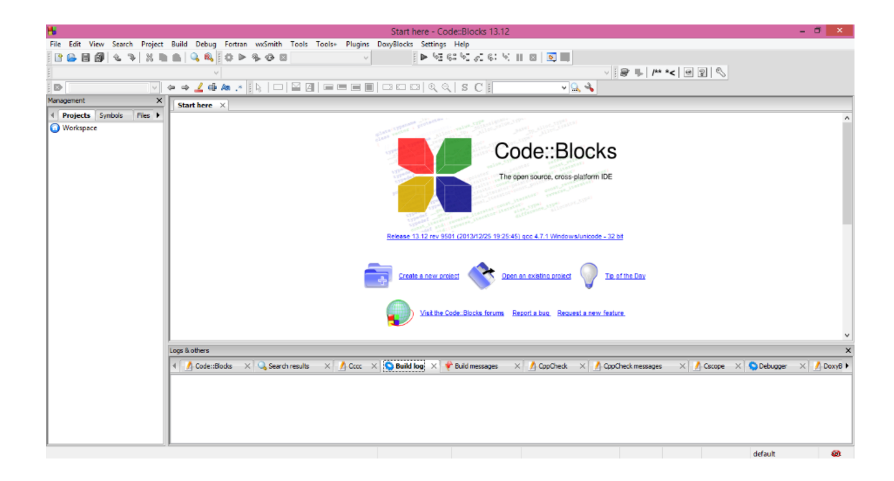
فى هذا الكتاب سنستخدم البيئة التطويرية Code::Blocks لتطوير البرامج بلغة السى ، وهذه البيئة متوفرة مجاناً يمكنك تحميلها من الرابط التالى :

http://sourceforge.net/projects/codeblocks/files/Binaries/13.12/Windows/codeblocks-13.12mingw-setup.exe/download

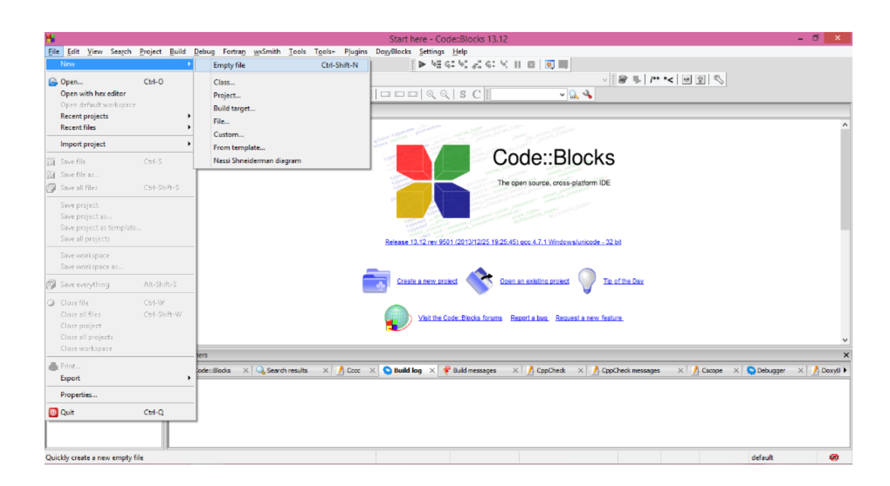
و هذه النسخة من البرنامج مصحوبة بمترجم GNU GCC Compiler. بعد تنزيل البرنامج قم بتنصيبه بكل سهولة و لا تغير شيئاً فى الإعدادات القياسية . و الآن سأتناول معكم كيفية التعامل مع البيئة التطويرية.

شرح التعامل مع البيئة التطويرية

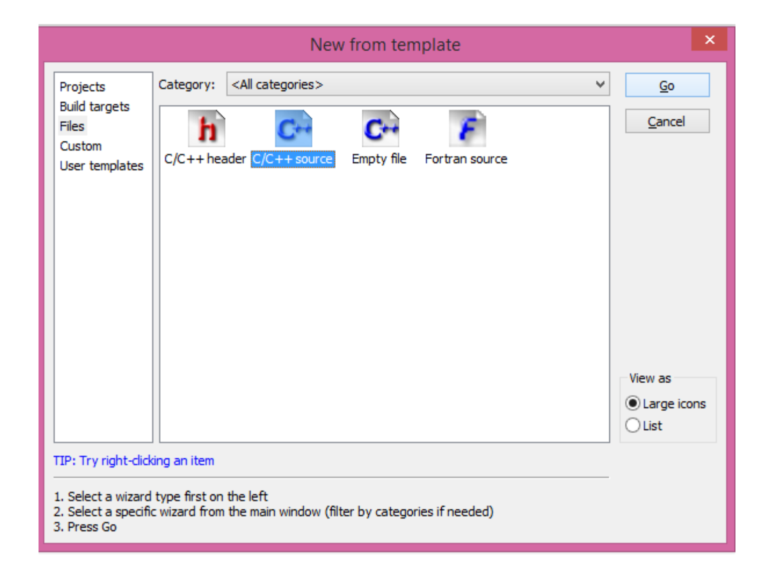
عند فتح البرنامج يسظهر لك هذه الشاشة الإفتتاحية



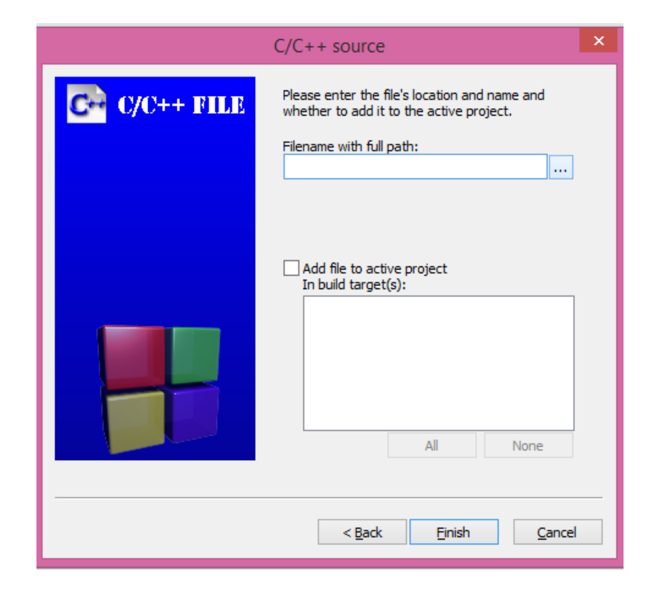
كما هو موضح ، من قائمة file ، اختر new ، ثم اختر new file .



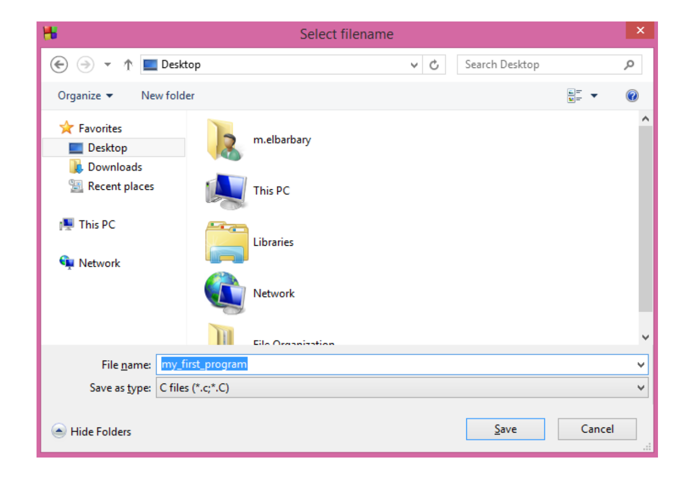
ثم اختر C/C++ Source ، ثم اضغط go.



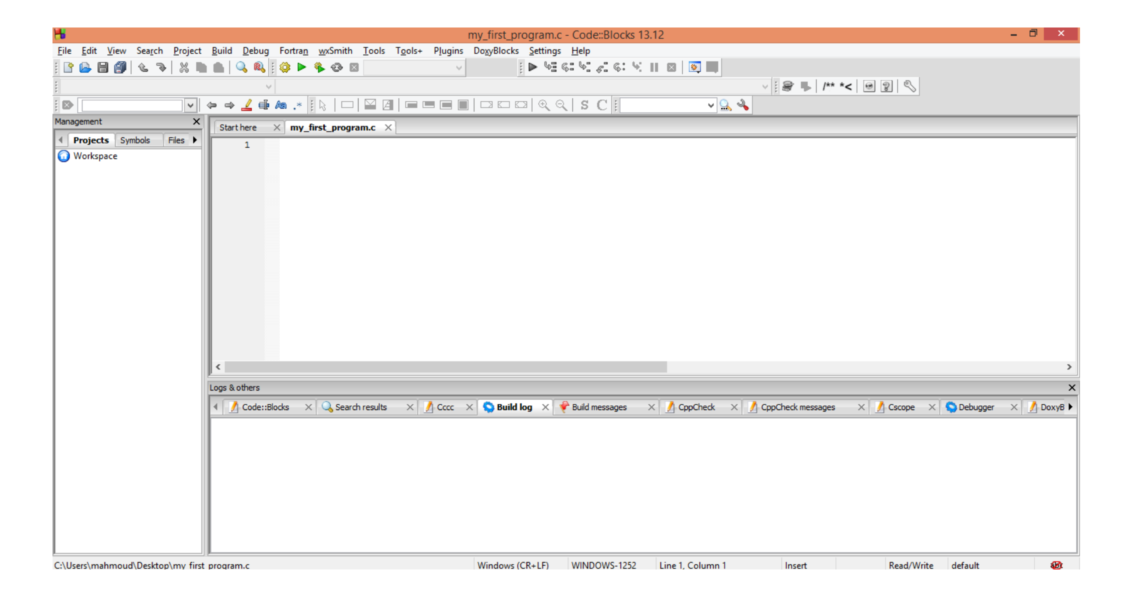
ثم اضغط على الزر المنقط الموضح بالصورة



قم بإختيار المكان الذى تريد حفظ الملف فيه، و اكتب الإسم الذى تريد تسمية البرنامج به، ثم اضغط Save .

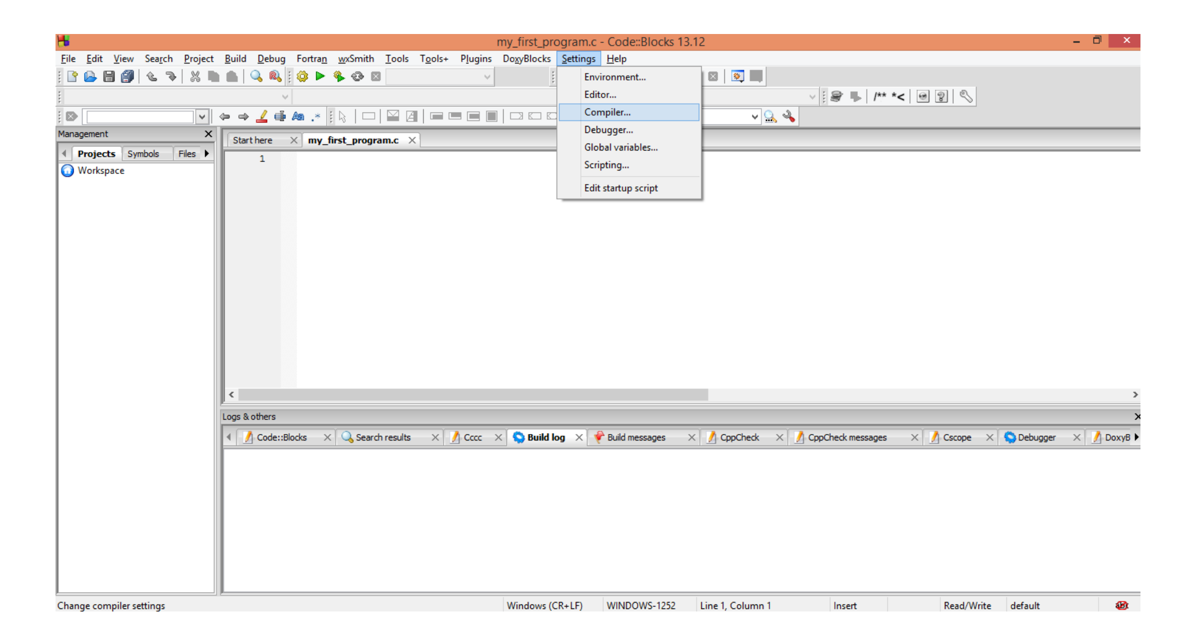


سيظهر لك هذه النافذة ، و هنا سنقوم بكتابة برامجنا الخاصة .

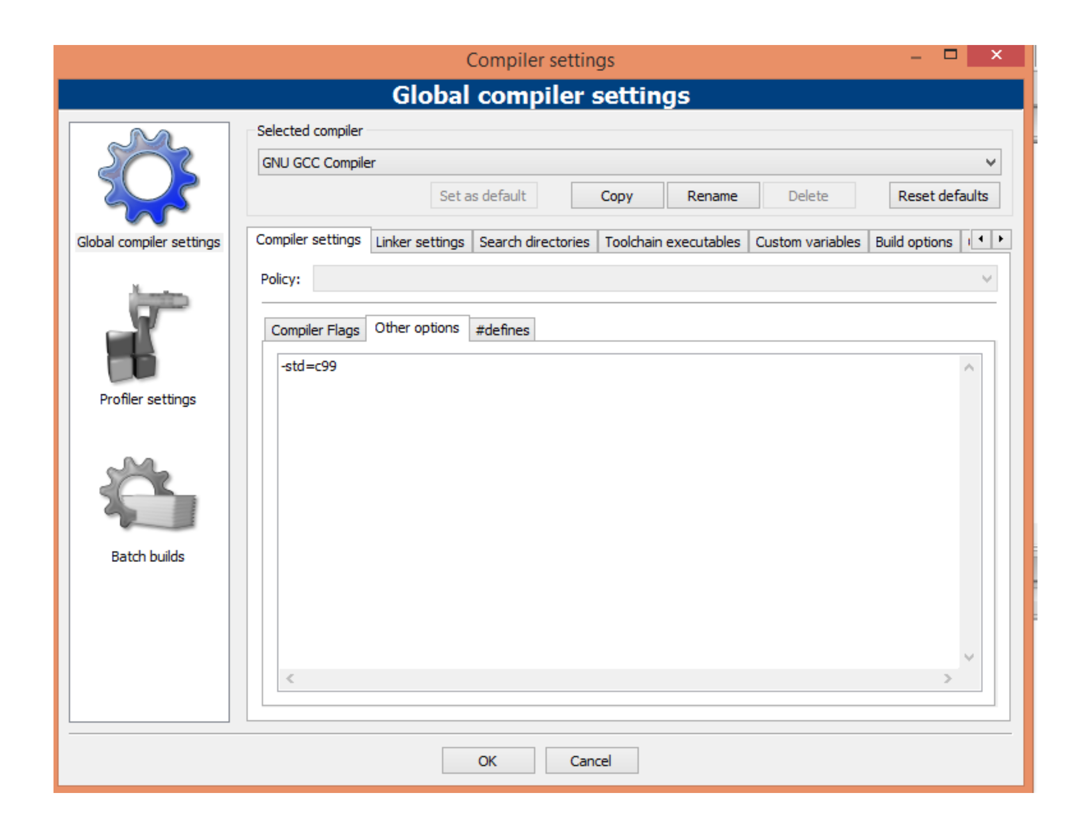


سنقوم بعمل برامجنا الخاصة فى النظام القياسى c99 ، و لتحويل عمل الـ compiler إلى نظام c99 ، نقوم بالآتى .

من قائمة settings ، نختار compiler .



ثم نقوم بإختيار other option ، ثم نقوم بكتابة std = c99- ، ثم نضغط على OK .



الآن تكون البيئة التطويرية جاهزة تماماً للبدء بإستخدامها فى برامجنا الخاصة .

برنامج

سنبدأ بكتابة برنامج بسيط فى البداية قبل أى شرح ، و ذلك لتألف البيئة التى تعمل بها و تستكشف بعض الأساسيات الثابتة فى أى برنامج .

```c
/*Your first program*/

#include <stdio.h>


int main()
{
    printf("My name is Mahmoud\n");
    return 0;
}
```

شرح البرنامج

```c
/*Your first program*/
```

يسمى كومنت أو تعليق ، و يمكن عمل كومنت فى لغة السى بإستخدام /* الكومنت المراد كتابته */ ، و الكومنت يستخدم لتوضيح أوامر البرنامج و أجزاءه ، و لا يؤثر على عمل البرنامج .

```c
#include <stdio.h>
```

تستخدم #include لإستيراد ملفات لداخل برنامجك،لإستخدام دوال منها ، و فى هذا المثال تم إستيراد ملف stdio.h من الـ standard library ، و يختص هذا الملف بدوال الإدخال و الإخراج مثل printf لطباعة خرج ، و scanf لإستقبال بيانات من المستخدم .

```c
int main()
```

هذه هى الدالة الرئيسية للبرنامج ، و يبدأ التنفيذ منها ، و أى برنامج يجب أن يحتوى عليها ، و يتم إحتواء الأوامر بداخلها بإستخدام {} ،و سيتم دراسة الدوال بشكل أكثر وضوحاً فى الباب المخصص لها.

```c
printf("My name is Mahmoud\n");
```

دالة printf تستخدم للطباعة ، و يتم وضع الجملة المراد طباعتها بين علامتين تنصيص كما فى المثال ، و سنتاولها لاحقاً بشىء من التفصيل.

```c
return 0;
```

هذه الجملة تخطر النظام المشغل بأن البرنامج تم تنفيذه بطريقة صحيحة دون حدوث أى مشاكل غير متوقعة.

هنا تكون قد إنتهيت من كتابة و فهم أول كود لك فى لغة السى، أتمنى أن تكون قد تحمست أكثر لمعرفة المزيد عن اللغة ، و نكون هنا قد انتهينا من الفصل الأول فى هذا الكتاب . سيكون حديثنا بإذن اللّه فى الفصل القادم عن المتغيرات .

الفصل الثانى

المتغيرات و العمليات الحسابية

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ ما هى أنواع المتغيرات فى لغة السى ؟
- ✓ كيفية تعريف المتغيرات و إعطائها قيم إبتدائية.
- ✓ العمليات الحسابية .
- ✓ طريقة إستخدام دالتى الإدخال و الإخراج scanf - printf .
- ✓ كيف يتم إستخدام الـ placeholders فى الإدخال و الإخراج .

أنواع المتغيرات

نبدأ مباشرة بالتعرف على أنواع المتغيرات فى لغة السى – من المفترض أن تكون على دراية بماهية المتغيرات سلفاً . تحتوى لغة السى على مجموعة من أنواع المتغيرات ، أهمها ما يلى .

الأرقام الصحيحة	الأرقام الغير صحيحة	حرف
int	float	char
long	double	
long long	long double	

اللغة لا تحتوى على نوع متغير نصى String ، و لكن يتم إستخدام مصفوفة من العناصر من النوع char ، و سنتناول معاً المصفوفات و المتغيرات النصية بالتفصيل فى الفصلين المخصصين لهما.

فى الحقيقة كل رقم يحتوى على " . " نقطة فهو يعتبر غير صحيح و إن كان صحيحاً فمثلاً 9.0 هذا الرقم يعتبر رقماً غير صحيحاً ، فالفارق بين الرقم الصحيح و الغير صحيح هو وجود النقطة ، متى وجدت كان العدد غير صحيحاً.

الفرق بين أنواع المتغيرات من النوع الواحد مثلا (int و long و long long) هى مساحة الذاكرة التى يتم حجزها لهذا المتغير ، ففى الغالب يشغل المتغير من النوع int مساحة 4 بايت من الذاكرة ، و long مساحة 8 بايت من الذاكرة ،و هذه الأرقام تعتمد على نوع النظام المشغل ، فهى تختلف من نظام مشغل إلى آخر.

تعريف المتغيرات

يتم تعريف المتغير عن طريق كتابة نوع المتغير أولا ثم إسم المتغير .هذا تعريف مجموعة مختلفة الأنواع من المتغيرات كمثال.

```
int x;
long long z;

float f;
double sa;

char cz;
```

يمكن إختيار أى إسم للمتغير الجديد الذى تقوم بتعريفه ، و لكن بشروط :

1. ألا يكون كلمة محجوز مثل int أو include مثلاً.
2. ألا يحتوى على رمز خاص مثال – ، \ ، / ، & إلخ ، و لكن يمكن إستخدام _ underscore .

أما إذا أردنا أن نقوم بتعريف متغير ثابت فإننا نستخدم كلمة const قبل التعريف ، أو عن طريق إستخدام #define ، كالآتى.

```
#define MAX 100

const int MAX = 100;
```

أى من الأمرين السابقين يقوم بتعريف متغير ثابت إسمه MAX من النوع int .

إعطاء قيمة إبتدائية لمتغير

فى المثال السابق على المتغيرات الثابتة ، قمنا بإعطائهم قيمة إبتدائية فى نفس أمر التعريف ، و قد نقوم بذلك بأكثر من طريقة ، الأمثلة الآتية للتوضيح.

```
int x = 0;
float sal = 1000.0f , model = 2.1f;
double bet = 12.5;
char code = 'c';
```

لاحظ أنه فى حالة إعطاء قيمة إبتدائية للمتغير من النوع float أو double ، يجب وضع " . " نقطة فى الرقم حتى و إن كان صحيحاً ، و يجب أن يوضع " f " فى نهاية القيمة من النوع float ، لكى لا يحدث مشاكل غير متوقعة عند إستخدام هذه القيمة فى عمليات حسابية فى البرنامج.

عملية نقل بيانات من متغير لآخر

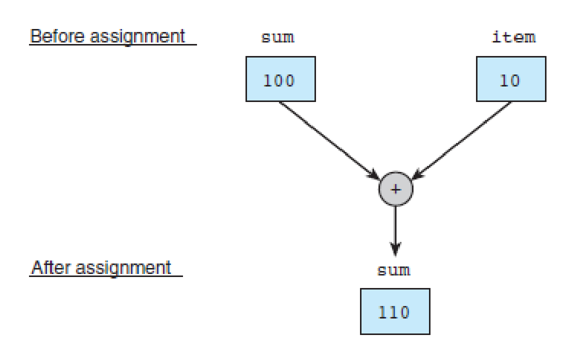
عملية الـ assignment هى عملية نقل البيانات من متغير لآخر ، أو نقل ناتج عملية حسابىة إلى متغير آخر .بصورة عامة فى أى عملية assignment، يتم نقل القيمة التى يعبر عنها الطرف الأيمن أياً كانت – سواء قيمة لمتغير أو ناتج عملية حسابية – إلى الطرف الأيسر .

فمثلاً إذا قمنا بالعملية الآتية :

```
int sum = 100 , item = 10;

sum = sum + item;
```

صورة توضيحية لنقل البيانات بعد عملية الجمع :



الآن و قد انتهينا من القواعد الأساسية الخاصة بالمتغيرات فى لغة السى ، نبدأ دراسة العمليات الحسابية ثم نتبعها بأمثلة مشروحة ، و تمارين عملية على المتغيرات و العمليات الحسابية .

العمليات الحسابية

تضمن لغة السى مجموعة من العمليات الرياضية التى نستخدمها بشكل مستمر فى برامجنا الخاصة ، و الجدول التالى يحتوى على العمليات المستخدمة :

الرمز	العملية
+	الجمع
-	الطرح
*	الضرب
/	القسمة
٪	باقى القسمة

نفترض أن لدينا متغيرين و ليكن i و j من النوع int ، و نريد إجراء جميع هذه العمليات الحسابية عليهم و حفظ ناتج كل عملية فى متغير جديد ، سنقوم بهذه العملية كالآتى .

```
int i ,j,sum,sub,div,mul,mod;

sum = i + j;
sub = i - j;
div = i / j;
mul = i * j;
mod = i % j;
```

عملية إيجاد باقى القسمة بإستخدام ٪ يجب أن يكون كلا طرفى العملية من int ، و لا يمكن إجراء هذه العملية على متغير من النوع float . و لكن يمكن التغلب على هذا ، بإستخدام ما يسمى الـ casting ، و هو عملية تحويل إجبارى من نوع إلى آخر .

مثال

```
6      float sum = 100.0f , item = 10.0f , res = 0.0;
7
8      res = sum % item ;
```

هنا تم إستخدام ٪ على متغيرين من النوع float ، لذلك سيظهر لك الخطأ الآتى .

```
error: invalid operands to binary % (have 'float' and 'float')
```

هنا يقوم المترجم (compiler) بإخطارك بوجود خطأ ، و هو إستخدام ٪ مع متغيرين من النوع float. للتغلب على هذه المشكلة يتم تحويل كلاً من المتغيرين إجبارياً إلى int عن طريق الـ casting ، كالآتى.

```
float sum = 100.0f , item = 10.0f , res = 0.0;

res = (int)sum % (int)item ;
```

لاحظ أنه فى عملية الـ casting سيتم إهمال أى كسر موجود فى المتغيرين .

عامل القسمة / يعمل بصورة طبيعية عند إستخدامه مع أعداد غير صحيحة ، أما إذا تم إستخدامه مع النوع int فإن الناتج لا بد و أن يكون int ، أى إنه يتم إهمال أى كسر ناتج عن العملية ، فمثلاً ناتج 2.0 / 5.0 يساوى 2.5 ، أما ناتج 5 / 2 يساوى 2 . لاحظ أنه تم إهمال الكسر. لذا عليك الإنتباه جيداً عند إجراء عملية القسمة على الأعداد الصحيحة لكى تحصل على نتائج سليمة.

هناك بعض الإختصارات للعمليات الحسابية ، فمثلاً يمكننا إستبدال هذه العملية

```
i = i +5;        i = i * (j/5);
```

بهذه

```
i += 5;          i *= j/5;
```

أو هذه العملية

```
i = i + 1;       i = i-1;
```

بهذه

```
i++; أو ++i;      i--; أو --i;
```

تعرف العملية الأخيرة بالـ increment و الـ decrement أى زيادة واحد على قيمة المتغير أو إنقاص واحد من قيمة المتغير.

دوال الإدخال و الإخراج

يمكن حفظ بيانات فى متغير معين فى الذاكرة بطريقتين ، الأولى أن يتم إعطاء هذا المتغير قيمة مباشرة عن طريق الـ assignment ، أو عن طريق إستقبال بيانات من المستخدم و حفظها فى هذا المتغير ، و نقوم بهذا عن طريق إستخدام دوال الإدخال و الإخراج ، و قد تعرضنا لواحدة منها سلفاً و هى دالة printf و قلنا أنها تستخدم لطباعة بيانات معينة للمستخدم ، و يوجد الكثير من دوال الإخراج على غرار دالة printf ، و كل هذه الدوال موجودة فى ملف (stdio.h) الذى تعرضنا له سابقاً، و سنستعرض الآن دالة printf و دالة scanf – تستخدم لعملية إدخال بيانات من المستخدم – بشىء من التفصيل .

دالة printf

إذا أردنا أن نقوم بعرض قيمة أى متغير للمستخدم أو جملة نصية ، فيجب علينا أن نستخدم printf ، كالمثال الآتى .

مثال

سنقوم بطباعة عمر شخص إسمه " على " بإستخدام printf .

```
#include <stdio.h>

int main(void){

    int age = 21;

    printf("Ali age = %d", age);

    return 0;
}
```

لاحظ تكوين جملة الطباعة

```
printf("Ali age = %d", age);
```

printf هى إسم الدالة ، و المعامل الأول دائماً يكون معامل نصى يوضع بين علامتى تنصيص مزدوج ، و يحتوى بداخله على ما يسمى بالـ placeholder ، أى النائب و سُمى كذلك لأنه ينوب عن المتغير الذى سيأتى فى المعامل الثانى ، فعند طباعة هذه الجملة للمستخدم ، ستظهر قيمة age مكان الـ placeholder ، فهو بذلك يحدد مكان وضع قيمة المتغير فى الجملة المطبوعة ، و هنا يوجد placeholder واحد ، لذلك أتى معامل واحد بعد المعامل النصى ، و إذا كان هناك إثنين placeholder فسيأتى معاملين بعد المعامل النصى يحتويان على قيم سيتم إستبدال الـ placeholder بهم و هكذا ، لذلك فأن الـ placeholder يحدد أيضاً عدد المعاملات أو المتغيرات التى ستأتى بعده .

خرج المثال

عند عمل run لهذا البرنامج ، سيكون الخرج كالآتى .

```
Ali age = 21
```

مثال

طباعة أعمار 3 أشخاص مختلفين .

```
#include <stdio.h>

int main(void){

    int age1 = 21,
        age2 = 14,
        age3 = 60;

    printf("Ali age = %d , Ahmed age = %d , Zaki age = %d",
           age1, age2, age3);

    return 0;
}
```

لاحظ أن الـ placeholder الأول سيتم التعويض عنه بقيمة age1 (أول معامل بعد المعامل النصى) ، و الـ placeholder الثانى سيتم التعويض عنه بقيمة age2 (ثانى معامل بعد المعامل النصى) ، و كذلك بالنسبة للثالث.

خرج البرنامج

فى هذه الحالة يكون خرج البرنامج كالآتى .

```
Ali age = 21 , Ahmed age = 14 , Zaki age = 60
```

placeholders

دائماً ما يبدأ الـ placeholder بـ ٪ ، و فى الأمثلة السابقة دائماً إستخدمنا ٪d لأنه كان ينوب عن متغير من النوع int ، و لكنه لا يكون ٪d فى كل الحالات ، إنما يتغير بتغير نوع المتغير ، و هذا جدول يوضح الـ placeholder الخاص بأكثر أنواع المتغيرات إستخداماً.

placeholder	نوع المتغير	الدالة المستخدم معها
%c	char	printf / scanf
%d	int	printf / scanf
%f	double	printf
%lf	double	scanf
%ld	long	printf / scanf
%f	float	printf / scanf

دالة scanf

هذه الدالة تستقبل البيانات المدخلة من المستخدم و تقوم بحفظها فى متغير ، كالمثال الآتى .

مثال

هذا البرنامج سيقوم بإستقبال رقم من المستخدم و حفظه فى متغير ، ثم طباعته مرة أخرى .

```
#include <stdio.h>

int main(void){

    int num;

    printf("Enter a number : ");
    scanf("%d", &num);

    printf("The number you have entered = %d", num);

    return 0;
}
```

```
scanf("%d", &num);
```

يتكون أمر إستقبال البيانات من اسم الدالة scanf ، و المعامل الأول مشابه للمعامل الأول فى دالة printf عبارة عن معامل نصى يحتوى على placeholder يحدد عدد و نوع المتغيرات التى سيتم إستقبال البيانات فيهم ، و المعامل الثانى هو المتغير الذى سيتم تخزين البيانات فيه.
لاحظ أنه تم وضع علامة & و تسمى address of operator أى العامل الذى يأتى بالعنوان ، و عند وضعه قبل متغير كما فى هذه الجملة فإنه يعنى أنه يقوم بإحضار عنوان هذا المتغير لإخبار دالة scanf بعنوان المتغير الذى اسمه num فى الذاكرة ليتم وضع القيمة المدخلة فيه .

خرج البرنامج

و يكون ناتج البرنامج السابق ، كالآتى.

```
Enter a number : 25
The number you have entered = 25
```

برنامج تطبيقى

هذه الفقرة تعد من أهم فقرات الكتاب ، و فيها يتم توظيف ما تم دراسته فى برنامج كبير نسبياً ، و هذا لتتعلم كيف توظف ما تعلمته فى عمل برامجك الخاصة ، كما يتم عرض المزيد من المعلومات الهامة خلال شرح البرنامج . و يفضل أن يتم فهم الكود جيداً ، ثم القيام بتنفيذه بنفسك دون الإستعانة بالكتاب إلا للضرورة القصوى .

اكتب برنامجاً يطلب من المستخدم إدخال قيمة مالية معينة ، ثم قم بعرض كيفية دفع هذا المبلغ عن طريق أقل عدد من الفواتير بقيمة 20 جنيه ،و 10 جنيهات ، و خمسه ، وواحد.فمثلاً إذا أدخل المستخدم مبلغ مالى بقيمة 112 جنيه ، يكون الخرج كالآتى :

20L.E bills = 5

10 L.E bills =1

5 L.E bills = 0

1 L.E bills = 2

```c
#include <stdio.h>

int main(){

    int amount , bill_of_20 , bill_of_10 ,
        bill_of_5 , bill_of_1;


    printf("Enter amount of money: ");
    scanf("%d" , &amount);

    bill_of_20 = amount / 20;
    amount %= 20;

    bill_of_10 = amount / 10;
    amount %= 10;

    bill_of_5 = amount / 5;
    amount %= 5;

    bill_of_1 = amount;

    printf("20 L.E bills = %d\n10 L.E bills = %d\n
           5  L.E bills = %d\n1  L.E bills = %d\n",
           bill_of_20 , bill_of_10 , bill_of_5 , bill_of_1);

    return 0;
}

```

شرح البرنامج

أولاً نقوم بتعريف المتغيرات التى سنستخدمها خلال البرنامج ، و فى هذا البرنامج سنستخدم 5 متغيرات واحدة للقيمة المدخلة ،و 4 لحفظ عدد كل نوع من الفواتير ، فنقوم بالآتى .

```c
int amount , bill_of_20 , bill_of_10 ,
    bill_of_5 , bill_of_1;
```

ثانياً نريد طلب إدخال قيمة المبلغ من المستخدم ثم إستقبالها فى متغير ، و يمكننا القيام بهذا عن طريق الأوامر الآتية .

```c
printf("Enter amount of money: ");
scanf("%d" , &amount);
```

و الآن نريد حساب أقل عدد من الفواتير المستخدمة لسداد هذا المبلغ ، أولاً يتم حساب عدد الفواتير ذات القمية 20 ، و ذلك بقسمة المبلغ على 20 ، و حفظ الناتج فى متغير من النوع int ، و هذا يعنى أنه سيهمل الباقى ، مثلا فى المثال المذكور ناتج 112/20 سيكون 5 فقط ، و بهذا نكون قد حسبنا عدد الفواتير ذات القيمة 20 التى نحتاجها .

الآن يجب الحصول على المتبقى من قسمة المبلغ على 20 ، و نقوم بذلك عن طريق التعبير الآتى 112٪20 يكون الناتج 12 ، نقوم بإستخدام الناتج فى حساب عدد الفواتير ذات القمية-10 ، و ذلك بقسمة المبلغ على 10 ، و حفظ الناتج فى متغير من النوع int ، و هكذا .. فيكون الكود المستخدم بهذا الشكل .

```
bill_of_20 = amount / 20;
amount %= 20;

bill_of_10 = amount / 10;
amount %= 10;

bill_of_5 = amount / 5;
amount %= 5;

bill_of_1 = amount;
```

الآن نقوم بطباعة عدد الفواتير من كل نوع للمستخدم بشكل مناسب و ذلك كالآتى .

```
printf("20 L.E bills = %d\n10 L.E bills = %d\n"
       "5  L.E bills = %d\n1  L.E bills = %d\n",
       bill_of_20 , bill_of_10 , bill_of_5 , bill_of_1);
```

لاحظ أنك إذا أردت كتابة جملة تريد طباعتها فى أكثر من سطر من الكود تقوم بإغلاق التنصيص على كل سطر على حدة و المترجم سيتخلص من علامات التنصيص أثناء الترجمة ،و لا يمكنك كتابتها بدون علامتى التنصيص فى نهاية الجملة الأولى أو بدونها فى بداية الجملة الثانية ، لأنه لن يعتبر أن الجملة الموجودة فى السطر الثانى جزء من الجملة الأولى .

و هنا أيضاً نلاحظ التعبير الآتى \n ، و هو يستخدم لبدء سطر جديد ، و تسمى هذه التعبيرات بالـ escape sequence ، و إليك أكثرها إستخدماً.

العمل	Escape sequence
بدء سطر جديد	/n
العودة إلى بداية السطر	/r
مسافة tab	/t
طباعة ما بعد الـ /	// , /' , /? , /"

و الآن نكون قد إنتهينا من تنفيذ أول برنامج تطبيقى فى هذا الكتاب ، عندما نتطرق لمواضيع أكثر فى دراستنا للغة ، ستكون البرامج التطبيقية لنا أكثر عملية . الآن أتركك مع التمارين.

التمارين

من أهم عوامل النجاح فى اتقان أى لغة برمجة، هو التدريب العملى المستمر على كتابة البرامج المختلفة، لذا سيكون كل موضوع فى هذا الكتاب مصحوباً فى نهايته بمجموعة جيدة من التمارين متدرجة الصعوبة، التى يجب أن تقوم بتنفيذها بنفسك .. أترككم مع أول تمرين.

1) اكتب برنامجاً يستقبل من المستخدم راتبه الأسبوعى بالجنيه المصرى ، و كذلك عدد ساعات عمله اليومية - كقيم من النوع float - ، ثم يقوم البرنامج بطباعة متوسط الأجر الذى يتقاضاه للساعة الواحدة على هيئة جنيهات و قروش .

2) إذا كان لديك نوعين من المنتجات ، الأول من فئة 3 جنيهات ، و الثانى من فئة 5 جنيهات ، اكتب برنامجاً يستقبل من المستخدم عدد القطع المراد شرائها من كل نوع ، ثم يقوم البرنامج بطباعة الحساب الإجمالى .

3) اكتب برنامجاً لحساب الوقت المستغرق فى تهذيب النبات بحديقة المنزل ، اطلب من المستخدم أبعاد المنزل علماً بأن المنزل على شكل مستطيل ، و كذلك نصف قطر الحديقة التى تحوى المنزل علماً بأنها على شكل دائرة ، علماً بأن المتر المربع الواحد يستغرق دقيقتان ، كم دقيقة تلزم لتهذيب حديقة المنزل.

الفصل الثالث

الجـــمـــل الشـــرطـــيـــة

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ كيف يتم إستخدام عوامل المقارنة فى بناء الشروط الخاصة بالجمل الشرطية ؟
- ✓ الجملة الشرطية if ، و أنواعها المختلفة .
- ✓ الجملة الشرطية switch case .

قبل البدء فى دراسة الجمل الشرطية ، ندرس فى البداية العوامل (operators) التى سنستخدمها فى تكوين الشرط الخاص بالجمل الشرطية ، و هناك نوعين من العوامل : عوامل المقارنة (comparison operators) و العوامل المنطقية (logic operators) .

عوامل المقارنة

العامل	المعنى
>	أكبر من
<	أصغر من
>=	أكبر من أو يساوى
<=	أصغر من أو يساوى
==	يساوى
!=	لا يساوى

انتبه (أخطاء شائعة) :

1) فى حالة اختبار تساوى قيمتين نستخدم == و ليس = .
2) إذا أردت اختبار وجود قيمة متغير بين قيمتين ، فالشرط يكتب هكذا مثلاً (1 < x && x > 5) ، و ليس على هذه الصورة (1 < x < 5) ، فهذه الصورة خاطئة.

العوامل المنطقية

المعامل	المعنى
!	يستخدم مع معامل واحد فقط ، إذا كان المعامل قيمته true يعود بـ false ، و العكس.
&&	يستخدم مع معاملين ، و يشترط تحققهما معاً - أى أن كلاهما true - لكى يقوم بتنفيذ جواب الشرط.
||	يستخدم مع معاملين ، و يشترط تحقق واحداً منهم على الأقل.

سيتبين لنا أكثر كيفية إستخدام تلك المعاملات من خلال الأمثلة القادمة .

جملة if الشرطية

بعد أن تناولنا أنواع العوامل المستخدمة فى بناء جملة الشرط ، نستعرض بناء أول نوع من أنواع الجمل الشرطية و هو if statement .

تتكون if فى أبسط صورها من شرط واحد و مجموعة من الأوامر يتم تنفيذهم عند تحقق هذا الشرط. كالآتى .

```
if(num > 0 && num < 5 )
    printf("num in range");
```

هذه الجملة الشرطية تقوم بطباعة الجملة الموضحة عندما تكون قيمة المتغير num أكبر من **0** و أقل من الـ **5** ، و if فى هذه الحالة تسمى simple if لأن هناك شرط واحد .

فى حالة وجود أكثر من أمر يُنفذ عند تحقق الشرط يتم إستخدام أقواس من النوع {} لتحتوى مجموعة الأوامر المراد تنفيذها عند تحقق الشرط. كالآتى.

```
if(num > 0 && num < 5 ){
    printf("num in range");
    printf("another order");
    x + y;
}
```

و يمكننا اختبار أكثر من شرط عن طريق if المتعددة الشروط كالآتى .

```
if(x == 5)
    printf("x = %d", x);

else if(x == 6)
    printf("x = %d", x);
```

و لكن ما الفرق بين الجملة الشرطية السابقة ، و هذه الجملة الشرطية ؟

```
if(x == 5)
    printf("x = %d", x);

if(x == 6)
    printf("x = %d", x);
```

الفرق أن البرنامج فى الحالة الأولى لا يختبر الشرط الثانى إذا تحقق الشرط الأول ، بينما فى الحالة الثانية يختبر الشرط الثانى سواء تحقق الأول أم لم يتحقق ، و هذا يبدو عقلانياً لأن الجملتين منفصلتين لا يؤثر تنفيذ أحدهما من عدمه على الآخر.

ويمكنك أن تقوم بتنفيذ مجموعة من الأوامر فى حال عدم تحقق أى من شروط الجلة الشرطية بإستخدام else كالآتى :

```
if(x == 5)
    printf("x = %d", x);

else if(x == 6)
    printf("x = %d", x);
else
    printf("ERROR");
```

فهذه الجملة تطبع كلمة ERROR فى حالة أن قيمة المتغير x لا تساوى 5 و لا تساوى 6 كذلك .

جملة if المتداخلة

يطلق عليها Nested if و هى عبارة عن جملة شرطية تحتوى بداخلها جملة شرطية أخرى أو أكثر ، كالمثال الآتى .

```
if(x == 5){

    printf("x = %d", x);

    if(y == 5){

        printf("y = %d", y);

    }
}
```

تنفذ جملة الطباعة الأولى عند تحقق الشرط الأول (x==5) ، و جملة الطباعة الثانية لا تنفذ إلا عند تحقق الشرط الأول (x==5) و الثانى (y==5).

مثال

برنامج يقوم بطباعة الرقم الأكبر من بين 3 أرقام يقوم بإدخالها المستخدم (بإستخدام nested if).

```
#include <stdio.h>

int main(void){

    int x ,y , z;

    printf("Enter 3 numbers (separate them with spaces) : ");
    scanf("%d%d%d", &x, &y, &z);

    if(x > y){
        if(x > z){
            printf("%d is the largest number", x);
        }else{
            printf("%d is the largest number", z);
        }
    }else{
        if(y > z){
            printf("%d is the largest number", y);
        }else{
            printf("%d is the largest number", z);
        }
    }

    return 0;

}
```

تعمدت هنا إستقبال البيانات كلها بإستخدام scanf واحدة ، لأبين لك أن دالة scanf دالة ذكية يمكنها استقبال أكثر من قيمة و تخزينها فى أكثر من متغير فى جملة واحدة ، على أن يفصل بين القيم المدخلة بمسافة أو مسافة tab أو enter ، فإذا أراد المستخدم إدخال معادلة فيجب إدخالها بالشكل التالى : 2 + 10 ، ثم يضغط Enter للإدخال ، على أن يحافظ على المسافات بين القيم المدخلة.

الجملة الشرطية الخارجية تختبر عما إذا كان x أكبر من y ، فى هذه الحالة هناك إحتمالين أن x هى أكبر الأرقام أو z هى الأكبر ، لذا استخدمنا جملة شرطية داخلية تختبر ما إذا كان x أكبر من z حينها تكون x هى الأكبر فيتم طباعتها للمستخدم ، أو غير ذلك (else) أى أن z أكبر من x و حينها يتم طباعة z ، أما اذا لم تكن x أكبر من y (else) للجملة الشرطية الخارجية ، فيوجد هنا احتمالين ، أن تكون y أكبر من z ، فيتم طباعة y ، أو أن z أكبر من y فيتم طباعة z .

Switch Case

و بعد أن تعرفنا على جملة if الشرطية ، نستعرض الآن ثانى أنواع الجمل الشرطية ، و هى جملة Switch case الشرطية ، شكلها البنائى العام كالآتى .

```
switch(المتغير){
    case القيمة الأولى:
        الأوامر المراد نفيذها عندما يساوى المتغير القيمة الأولى
        break;
    case القيمة الثانية:
        الأوامر المراد نفيذها عندما يساوى المتغير القيمة الثانية
        break;
      .
      .
      و هكذا
      .
      .
     default:
          الأوامر المراد تنفيذها عند عدم تحقق أى من الشروط
}
```

لاحظ أن :

1) تعمل default عمل else فى جملة if .
2) تستخدم switch case مع النوع char و int فقط ، و لا تعمل مع النوع double .
3) لا يمكن إختبار مدى معين بإستخدام switch case ، أى لا يمكن أن نقول case x<5: .
4) لاحظ أنه يجب وضع break فى آخر كل حالة ، ليتم الخروج من جملة Switch case كاملة بعد تنفيذ الأوامر ، و لا يجب وضعها فى آخر حالة الـ default ، لأن آخر الجمل و التى سيتم الخروج من Switch case بعد الإنتهاء منها.

مثال

فى هذا المثال نستقبل رقماً من المستخدم ، و على حسب الرقم نطبع جملة معينة .

```
#include <stdio.h>

int main(){

    int x;

    printf("Enter a number from 1 to 3: ");
    scanf("%d", &x);

    switch(x){

        case 1:
            printf("good morning!");
            break;

        case 2:
            printf("good afternoon!");
            break;

        case 3:
            printf("good night!");
            break;

        default:
            printf("This number is not allowed.");
    }

    return 0;

}
```

فى أغلب الحالات يتم إستخدام if فى بناء الجمل الشرطية ، و لكن هناك حالات قليلة يفضل استخدام switch case مثلاً إذا كان المتغير الذى نختبر عليه الشروط له مدى محدود من القيم (10 قيم أو أقل) و يكون من النوع char أو int .

برنامج تطبيقى

برنامج آلة الحاسبة ، يقوم بالعمليات التالية (الجمع ، الطرح ، الضرب ، القسمة ، باقى القسمة) ، على أن تستقبل من المستخدم العملية التى يريد القيام بها و كذلك العددين المراد إجراء العملية عليهما .

```
1   #include <stdio.h>
2
3
4   int main(){
5
6       float num1,
7             num2,
8             res;
9
10      char op;
11
12      printf("Enter the arithmetic equation \n");
13      scanf("%f %c %f", &num1, &op, &num2);
15      if(op == '+')
16          res = num1 + num2;
17      else if(op == '-')
18          res = num1 - num2;
19      else if(op == '*')
20          res = num1 * num2;
21      else if(op == '/')
22          res = num1 / num2;
23      else if(op == '%')
24          res = (int)num1 % (int)num2;
25
26      printf("result = %.2f", res);
27
28      return 0;
29  }
```

شرح البرنامج

أولاً نقوم بتعريف المتغيرات التى سنستخدمها خلال البرنامج ، 3 متغيرات من النوع float لتخزين العددين ، و الناتج. و متغير من النوع char لتخزين نوع العملية.

```
float num1,
      num2,
      res;

char op;
```

ثم نقوم بطلب و إستقبال البيانات المطلوبة من المستخدم .

```
printf("Enter the arithmetic equation \n");
scanf("%f %c %f", &num1, &op, &num2);
```

ثم قمنا هنا بإختبار نوع العملية المطلوبة، وبناءاً على طلب المستخدم نقوم بإجراء العملية الحسابية المناسبة. لاحظ أننا قمنا بتحويل كلاً من العددين إلى int عندما إستخدمنا المعامل ٪ لأنه لا يستخدم على النوع float ، ، ثم نقوم فى النهاية بعرض ناتج العملية .

```
    if(op == '+')
        res = num1 + num2;
    else if(op == '-')
        res = num1 - num2;
    else if(op == '*')
        res = num1 * num2;
    else if(op == '/')
        res = num1 / num2;
    else if(op == '%')
        res = (int)num1 % (int)num2;

    printf("result = %.2f", res);

    return 0;
}
```

هنا نكون قد انتهينا من شرح الجمل الشرطية فى لغة السى ، يجب أن تكون قد تعلمت كيفية إستخدام الجمل الشرطية فى أبسط الصور ، و من خلال الأمثلة و البرامج فى المواضيع القادمة التى بالتأكيد سنستخدم فيها الكثير من الجمل الشرطية ، ستألف أكثر كيفية العمل معها و كيفية توظيفها فى برامجك الخاصة بمنتهى السهولة .. أتركك الآن مع التمارين .

تمارين

1) اكتب برنامجاً لحساب قيمة المشتريات من منتج معين قيمة القطعة الواحدة منه 10 جنيهات، و يوجد تخفيض على أى كمية اكثر من 50 قطعة يبلغ 10٪ و على أى كمية أكثر من 100 قطعة يبلغ 20٪، لا تنسى، استقبل الكمية المراد شرائها من المستخدم ثم قم بطباعة قيمة المشتريات بعد التخفيض المناسب.

2) اكتب برنامجاً يقوم بإستقبال درجة الطالب فى صورة رقمية ثم يطبعها فى صورة حرفية (حيث : A = 85 - 100 ، B = 85 – 75 ، C = 75 – 65 ، D = 65 – 50 ، F = 0 – 50).

3) اكتب برنامج آلة حاسبة بإستخدام Switch case .

4) اكتب برنامجاً يستقبل من المستخدم نقطة على المستوى x-y ، ثم يقوم بطباعة عما إذا كانت النقطة تقع على أحد المحورين أم تقع فى ربع من الأرباع الأربعة و فى أى ربع تقع .

5) اكتب برنامجاً يقوم بإستقبال التاريخ من المستخدم على هيئة 3 أرقام صحيحة، الأول اليوم، و الثانى الشهر،و الثالث السنة. ثم يقوم بطباعة هذا التاريخ على هذه الهيئة 8^{th} October 1993 إذا كان الدخل 8 10 1993 . (لاحظ 1^{st} , 2^{nd} , 3^{rd} , 21^{st} , 22^{nd} , 23^{rd} و الباقى يأخذ th مثل 4^{th} , 5^{th} و هكذا.

الفصل الرابع

الحـــلـــقـــات الـــتـــكـــراريـــة

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ الحلقة التكرارية while .
- ✓ الحلقة التكرارية do-while .
- ✓ الحلقة التكرارية for .
- ✓ الأمرين break و continue .

لماذا نستخدم الحلقات التكرارية ؟

إذا أردت أن تقوم بتنفيذ أمر معين و ليكن أمر طباعة جملة معينة 10 مرات ، ماذا ستفعل ؟ .. ستقوم بكتابة 10 جمل طباعة ؟ .. إذا أردت طباعتها 100 مرة ؟ .. سيصبح الأمر سخيفاً إذا استمريت فى كتابة جملة الطباعة هذا العدد الكبير من المرات. الحلقات التكرارية يمكنها أن تتعامل مع هذه المشكلة ، فهى تقوم بتكرار مجموعة معينة من الأوامر أكثر من مرة سواء عدد محدد من المرات أو تكرارها حتى وقوع حدث معين يتوقف التكرار عنده .

الحلقة التكرارية while

هى أول نوع من الحلقات التكرارية التى سنقوم بشرحها فى هذا الفصل ، و بناؤها العام كالآتى.

```
while(الشرط){
    الأوامر المراد تنفيذها
}
```

فى حالة تحقق الشرط سيستمر تكرار تنفيذ الأوامر التى بداخلها ، حتى ينتفى الشرط فتتوقف عملية التكرار و يُستكمل تنفيذ البرنامج من بعد الحلقة التكرارية.

مثال

```
int i = 0;
while(i < 10){
    printf("%d", i);
    i++;
}
```

فى هذه الحلقة التكرارية يتم إختبار الشرط 10 > i فى كل مرة ليتم تنفيذ جملة الطباعة ، و فى البداية تكون قيمة 0 = i و هذا يعنى تحقق الشرط ، فيتم تنفيذ جملة الطباعة ثم تزداد قيمة i بمقدار واحد بناءاً على الأمر ++i (العداد) ، ثم تبدأ الحلقة من جديد من إختبار الشرط وحينها سيتحقق الشرط لأن i = 1 أى أنها أقل من

10 فيتم تنفيذ جملة الطباعة و هكذا حتى تكون i = 10 و حينها لا يتحقق الشرط فلا يتم تنفيذ الأوامر الموجودة بداخل الحلقة .

خرج المثال

```
0123456789
Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

لاحظ أن العداد الخاص بـ while loop يكتب مع باقى الأوامر بداخل الحلقة ، و لا حظ أيضاً أن تعريف المتغير i المستخدم فى الحلقة تم تعريفه بخارجها قبل البدء فى الحلقة التكرارية.

الحلقة التكرارية do while

و بعد أن تعرفنا على while loop نتعرف الآن على نوع خاص منها وهو do-while loop ، و تتخذ فى بنائها العام الشكل الآتى.

```
do{
    الأوامر المراد تنفيذها
}while(الشرط);
```

هذه الحلقة تقوم بنفس ما تقوم به حلقة while loop ، و لكنها تقوم بتنفيذ الأمر مرة واحدة قبل إختبار الشرط ، و انتبه إلى وجود ; بعد الشرط لأنها غير موجودة فى while loop .

تستخدم do-while loop نادراً مقارنة بـ while loop و for loop ، و أشهر إستخدام لهذه الحلقة التكرارية هو إستخدامها عندما تريد أن تكرر مجموعة من الأوامر على الأقل مرة واحدة.

مثال

نفس المثال السابق و لكن بإستخدام do while

```
int i = 0;

do{

    printf("%d", i);
    i++;

}while(i < 10);
```

الحلقة التكرارية for

يفضل كثير من المبرمجين إستخدام for loop فى اغلب الأحيان ، لأنه ببساطة يتم تعريف المتغير المستخدم فى الحلقة و تحديد الشرط و العداد فى سطر واحد فقط ، و هو ما يسهل كثيراً على المبرمج. و هذا هو البناء العام لـ for loop .

```
for(تعريف المتغير; الشرط; العداد){

    الأوامر المراد تنفيذها
}
```

مثال

نفس المثالين السابقين و لكن بإستخدام for

```
for(int i = 0; i < 10; i++)
    printf("%d", i);
```

هذه الحلقة أيضاً تقوم بما قامت به الحلقات السابقة ،و لكن لاحظ التكوين المختلف تماماً لـ for loop ، و لاحظ بساطة تركيبها. و تعمدت هنا عدم وضع {} لأذكرك إنه يمكن عدم وضعها فى حال كانت الأوامر المراد تنفيذها أمر واحد سواء فى جملة if الشرطية أو فى أى من الحلقات التكرارية.

يمكن الإستغناء عن أى من المعاملات الثلاثة (تعريف المتغير و الشرط و العداد) أو يمكن الإستغناء عنهم جميعاً و تركهم فارغين إذا ما أردت ذلك ، و لن ينتج هذا عن أى أخطاء و لكن يجب أن يكون بالشكل التالى .

```
for(; ;){
    الأوامر المراد تنفيذها
}
```

الأمران break و continue

يستخدم هذان الأمران فى أغلب الأحيان مع الحلقات التكرارية ، و يختلف عمل أحدهما عن الآخر .

الأمر break

يقوم هذا الأمر بالخروج من الحلقة التكرارية فوراً ، و غالباً ما يتم إستخدام شرط معين إذا تم تحققه ، يتم تنفيذ الأمر break و الخروج من الحلقة التكرارية .

مثال

```
int main(void){

    for(int i = 0; i < 10; i++){
        if(i == 5)
            break;
        printf("%d", i);
    }

    return 0;
}
```

من المفترض أن هذا البرنامج يقوم بطباعة الأعداد الصحيحة من 1 إلى 9 ، و لكننا قمنا بإدخال جملة شرطية تقوم بتفيذ أمر break عندما تكون i تساوى 5 ، و سيتم الخروج من الحلقة التكرارية تماماً فى الحال ، فلا يتم تنفيذ جملة الطباعة التى ستقوم بطباعة رقم 5 ، و ما بعدها من تكرارات .

خرج المثال

فيكون خرج المثال السابق كالآتى .

```
01234
Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

الأمر continue

يقوم الأمر continue عند تنفيذه بعدم تنفيذ ما تبقى من أوامر الحلقة التكرارية الحالية فقط ، و يقوم بتنفيذ باقى الحلقات التى تليها بصورة طبيعية .

مثال

نفس المثال السابق و لكن تم إستبدال الأمر break بالأمر continue .

```
int main(void){

    for(int i = 0; i < 10; i++){
        if(i == 5)
            continue;
        printf("%d", i);
    }

    return 0;
}
```

فى هذه الحالة عندما تكون i تساوى 5 ، سيقوم البرنامج بتنفيذ الأمر continue ، و سيتم التغاضى عن أى أوامر تأتى بعدها – جملة الطباعة التى تقوم بطباعة الرقم 5 - و لكن ستكمل الحلقة التكرارية عملها بشكل طبيعى بعدها فيتم طباعة رقم 6 و 7 و 8 و 9 .

خرج المثال

```
012346789
Process returned 0 (0x0)   execution time : 0.011 s
Press any key to continue.
```

لاحظ هنا أنه لم يتم طباعة الرقم 5 .

برنامج تطبيقى

سنقوم بتصميم لعبة تخمين ، بمعنى أنى سأختار رقماً فى مدى محدد ، و على المستخدم تخمين هذا الرقم بصورة صحيحة و لديه 3 محاولات متاحة فقط عليه أن يخمن الرقم صحيحاً قبل نفاذ محاولاته.

```
#include <stdio.h>

int main(){



    int chosen_num = 4,
        guessed_num;

    printf("WELCOME TO GUESSING GAME !!\n"
           "---------------------------\n");
    printf("I have chosen a number from 1 to 20 and You"
           " have just\n3 tries to guess it right.Let\'s start\n");
    printf("---------------------------\n");


    for(int tries = 3; tries > 0; tries--){

        printf("You have %d %s remaining\n",tries,
               tries == 1 ? "try" : "tries" );
        printf("Enter the number: ");
        scanf("%d", &guessed_num);

        if(guessed_num == chosen_num){
            printf("You have guessed it right.You Win !!");
            return 0;
        }else{
            if(guessed_num <= 20 && guessed_num > 0 )
                printf("You have guessed it wrong.Try again!\n");
            else
                printf("You have entered an out of range number"
                       ".Try again!\n");
        }
    }

    printf("You have finished your 3 tries and can\'t guess "
           "the number. You Lose !!");

    return 0;
}
```

الشرح

نقوم بتعريف المتغيرات التى سنستخدمها .

```
int chosen_num = 4,
    guessed_num;
```

نعرف متغير للرقم الذى نختاره و نعطيه قيمة إبتدائية و ليكن 4 ، و متغير لإستقبال الرقم الذى سيقوم بتخمينه المستخدم.

نقوم برسم الشكل العام للبرنامج ، و طباعة جمل للمستخدم توضح فكرة اللعبة ، كالآتى.

```
printf("WELCOME TO GUESSING GAME !!\n"
       "---------------------------\n");
printf("I have chosen a number from 1 to 20 and You have just\n"
       "3 tries to guess it right.Let\'s start\n");
printf("---------------------------\n");
```

ثم نقوم بعمل for loop تتيح للمستخدم 3 محاولات فقط ،و يتم إختبار عما إذا كان الرقم المدخل صحيحاً أم خطأ ، و إذا كان خطأ هل هو بداخل المدى المحدد(20 – 1) أم خارجه، كالآتى .

```
for(int tries = 3; tries > 0; tries--){

    printf("You have %d %s remaining\n",tries,
           tries == 1 ? "try" : "tries" );
    printf("Enter the number: ");
    scanf("%d", &guessed_num);

    if(guessed_num == chosen_num){
        printf("You have guessed it right.You Win !!");
        return 0;
    }else{
        if(guessed_num <= 20 && guessed_num > 0 )
            printf("You have guessed it wrong.Try again!\n");
        else
            printf("You have entered an out of range number"
                   ".Try again!\n");
    }
}
```

فنلاحظ هنا أننا أعطينا لعدد المحاولات قيمة إبتدائية 3 و العداد يقل كل مرة واحد و الشرط أن المحاولات لا تقل عن 1 ، أى أن الحلقة ستكرر نفسها 3 مرات كحد أقصى.فى بداية كل حلقة، نقوم بطباعة عدد المحاولات المتبقية و نطلب من المستخدم إدخال الرقم ثم نستقبل الرقم المدخل.

جملة الطباعة الأولى ، محدد النوع الثانى بها s٪ أى ان المعامل الثانى القادم سيكون string ، و لكن ما هذه الجملة "tries" :"try" ? tries == 1 هذه تساوى تماماً if (tries == 1) "try" else "tries" ، أى أنه إذا كانت عدد المحاولات واحدة فيتم طباعة try و إذا كان غير ذلك أى أكبر فأطبع tries ، فهذه الجملة التى رأيتها مجرد اختصار لـ if .

ثم قمنا بإختبار عما إذا كان الرقم صحيحاً أم لا ، إذا كان صحيحاً نقوم بطباعة جملة تخبر المستخدم بأن تخمينه كان صحيحاً ثم ننهى البرنامج عن طريق الامر return 0 ، أما إذا كان خطأ فنختبره عما إذا كان فى المدى من 1 إلى 20 أم لا . إذا كان فى المدى، أخبرنا المستخدم أن اختياره غير صحيح، و إذا كان فى غير المدى أخبرناه أن الرقم الذى خمنه خارج المدى.

فإذا انتهت الثلاث محاولات و لم يأتى المستخدم بالرقم الصحيح تنتهى الـ for loop ، فنطبع بعد الـ loop جملة تخبر المستخدم أنه قد استنفذ محاولاته الثلاثة، كالآتى.

```
printf("You have finished your 3 tries and can\'t guess "
       "the number. You Lose !!");
```

اختبار البرنامج

و الآن نجرب البرنامج لنتأكد إنه يعمل بالشكل الصحيح، ندخل له قيمة خاطئة فى داخل المدى و أخرى خارجه وأخيرة صحيحة. فيكون الخرج كما هو متوقع كالآتى.

```
WELCOME TO GUESSING GAME !!
---------------------------
I have chosen a number from 1 to 20 and You have just
3 tries to guess it right.Let's start
---------------------------
You have 3 tries remaining
Enter the number: 19
You have guessed it wrong.Try again!
You have 2 tries remaining
Enter the number: 22
You have entered an out of range number.Try again!
You have 1 try remaining
Enter the number: 4
You have guessed it right.You Win !!
Process returned 0 (0x0)   execution time : 12.485 s
Press any key to continue.
```

تمارين

1) اكتب برنامجاً يقوم بحساب القيمة الكبرى و القيمة الصغرى من بين مجموعة من الأرقام يقوم بإدخالها المستخدم ،و يقوم بطباعة مدى هذه القيم. استقبل عدد تلك القيم المدخلة من المستخدم أولاً.

2) اكتب برنامجاً يقوم بحساب متوسط مجموعة من الأرقام يتم إستقبالها من المستخدم، و كذلك مجموع مربعات هذه القيم، و حساب الإنحراف المعيارى لهم .علماً بأن الإنحراف المعيارى يساوى جذر «مجموع المربعات مقسوماً على عدد القيم ثم مطروحاً من مربع المتوسط». اسعتن بـ math.h لحساب الجذر بإستخدام دالة sqrt().

3) اكتب برنامجاً يقوم بحساب القاسم المشترك الأكبر بين رقمين يقوم المستخدم بإدخالهم .

4) عدل برنامج لعبة التخمين ، بحيث يتيح للمستخدم اختيار عما إذا كان يريد أن يلعب مرة أخرى بعد إنتهاء اللعبة أم لا ، إذا اختار أن يلعب مرة أخرى يجب أن تبدأ اللعبة فى العمل من جديد.

5) اكتب برنامجاً يقوم بطباعة أول 50 عدد فى متتابعة فيبوناتشى ، علماً بأن هذه المتتابعة الحسابية يتكون فيها كل عدد من مجموع العددين السابقين له ، و أول و ثانى رقم فى السلسلة يساوى 1 . (1,1,2,3,5,8....) و هكذا.

الفصل الخامس

المـــــــصـــــــــفـــــــــوفـــــات

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ ما هى المصفوفات ، و لماذا يتم إستخدامها .
- ✓ كيفية تعريف المصفوفات .
- ✓ المصفوفات الثنائية .
- ✓ أشهر العمليات على المصفوفات.

لماذا يتم إستخدام المصفوفات ؟

إذا أردت إستخدام 3 متغيرات مثلاً من النوع int ، ماذا ستفعل ؟ .. ستقوم بتعريف كل واحد منهم على حدة بالطريقة العادية . نفرض أنك أردت أن تقوم بتعريف 100 متغير ؟ .. هنا يصبح الأمر شبه مستحيل ، لذلك يتم إستخدام المصفوفات لتحتوى بداخلها مجموعة من العناصر من نفس النوع .

ما هى المصفوفات ؟

المصفوفات هى أشهر أنواع هياكل البيانات (data structure) – و هى مجموعة من البيانات تجمعها صفة معينة - ، و الصفة التى تجمع عناصر المصفوفة هى أنهم من نفس النوع .

تعريف المصفوفات

يتم تعريف المصفوفة كأى متغير آخر ، مع زيادة قوسين من النوع [ ] بعد إسم المتغير و بداخله يتم وضع عدد عناصر تلك المصفوفة. فمثلاً يتم تعريف مصفوفة من النوع int عدد عناصرها 10 كالآتى.

```
int x[10];
```

و يمكننا أن نضع لهذه المصفوفة القيم الإبتدائية التى تحملها ، و لاحظ أنه يمكن الإستغناء عن وضع عدد عناصر المصفوفة إذا تم وضع لها قيم إبتدائية أثناء التعريف. كما بالشكل الآتى .

```
int x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

و يمكنك تخيل شكل مصفوفة فى الذاكرة - عناصرها {1, 2 , 4, 8 , 16} - كما بالشكل.

1	a[0]
2	a[1]
4	a[2]
8	a[3]
16	a[4]

لاحظ أن رتبة العناصر تبدأ من صفر و ليس 1 ، لذلك كان رتبة آخر عنصر أقل من عدد عناصر المصفوفة بواحد، و يمكننا الوصول لأى عنصر فى المصفوفة عن طريق رتبته ثم استخدامه فى أى عملية مثله مثل أى متغير آخر تعاملنا معه من قبل ، فمثلاً لو أردنا أن نجمع العنصر الأول و الأخير فى هذه المصفوفة و نحفظهم فى متغير آخر ، سنقوم بذلك كالآتى.

```
y = x[0] + x[9];
```

يطلق على المصفوفات فى الحالات السابقة أحادية البعد ، و لكن هناك مصفوفات ذات بعدين أو ثلاثة أو أكثر، و يتم تعريفهم و إعطائهم قيم إبتدائية كما سنرى ذلك فى مثال مع المصفوفات ذات البعدين. عدد الصفوف يكتب فى الأقواس [] الأولى ،و عدد الأعمدة يكتب فى الأقواس الثانية.

```
int x[3][3] = {{1, 2, 3},
               {4, 5, 6},
               {7, 8, 9}};
```

المصفوفة التالية مصفوفة 3 * 4 ، أى مكونة من 3 صفوف و 4 أعمدة ،و يمكنك تخيل شكلها فى الذاكرة كالآتى.

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

و لكن ماذا لو أردنا إجراء عمليات معينة على عناصر المصفوفة كلها ، مثلاً إذا أردنا أن نطبع كل عناصر المصفوفة أو نجمعها كلها أو نقوم بترتيبها ؟ .. سنقوم بذلك عن طريق إستخدام الحلقات التكرارية .

مثال

طباعة جميع عناصر المصفوفة.

```
int x[3] = {1, 2, 3};

for(int i = 0; i < 3; i++)
    printf("%d", x[i]);
```

مثال

إستقبال عناصر المصفوفة من المستخدم.

```
int x[3];

printf("ENTER 3 VALUES: ");

for(int i = 0; i < 3; i++)
    scanf("%d", &x[i]);
```

مثال

نقوم بإستقبال عناصر المصفوفة من المستخدم، ثم نحصل على مجموع عناصر المصفوفة، ثم طباعة الناتج.

```
int x[5],
    sum = 0;

printf("ENTER 5 VALUES: ");

for(int i = 0; i < 5; i++){
    scanf("%d", &x[i]);
    sum += x[i];
}

printf("sum = %d", sum);
```

و من أشهر العمليات على المصفوفات البحث عن رتبة عنصر له قيمة معينة ، و كذلك ترتيب عناصر المصفوفات ، و سنتعرض لكلا النوعين من العمليات لأهميتهم .

مثال

البحث عن رتبة عنصر ذو قيمة معينة ، نوضح كيفية القيام بتلك العملية بالمثال الآتى.

```
int x[5];

printf("ENTER 5 VALUES: ");

for(int i = 0; i < 5; i++)
    scanf("%d", &x[i]);

for(int i = 0; i < 5; i++){
    if(x[i] == 1)
        printf("the index of value 1 = %d", i);
}
```

مثال

ترتيب عناصر المصفوفة ، و فى هذا المثال نقوم بترتيب عناصر المصفوفة تصاعدياً.

```
int x[5],
    med;

printf("ENTER 5 VALUES: ");

for(int i = 0; i < 5; i++)
    scanf("%d", &x[i]);

for(int i = 0; i < 4; i++){
    for(int j = i + 1; j < 5; j++){
        if(x[i] > x[j]){
            med = x[i];
            x[i] = x[j];
            x[j] = med;
        }
    }
}

for(int i = 0; i < 5; i++)
    printf("\n%d", x[i]);
```

فى هذا المثال يتم مقارنة كل عنصر فى المصفوفة إبتداءاً من العنصر الأول بما يليه من العناصر ، فإذا كان أحد تلك العناصر أصغر منه يتم تبديل قيمتى العنصرين ليصبح الأصغر هو الأول فى الترتيب ، ثم الانتقال إلى العنصر الثانى فى المصفوفة و مقارنته بما بعده و هكذا. فنجد فى الحلقتين التكراريتين ، الحلقة الأولى تقوم بالمرور على كل عنصر فى المصفوفة لتضعه تحت الإختبار، و الثانية تمر على كل عنصر بعده لتختبره بالنسبة لهذا العنصر ، و لاحظ أنه تم إختبار جميع عناصر المصفوفة عدا الأخير ، لأنه لا يوجد عناصر بعده .

برنامج تطبيقى

تصميم لعبة « x – o » يشترك فيها لاعبان.

```
1  #include <stdio.h>
2
3  int main(){
4
5      char x[3][3] = {{'1','2','3'},
6                      {'4','5','6'},
7                      {'7','8','9'}};
8
9      int plays = 1,
10         slot,
11         row,
12         column;
13
14     printf("Welcome to X - O game\n");
15     printf("---------------------\n\n");
16
17     for(int i = 0; i < 3; i++){
18         printf("     ");
19         for(int j =0; j < 3; j++)
20             printf("%c | ", x[i][j]);
21     printf("\n     -----------\n");
22     }
23
24     do{
25
26     printf("\nPlayer %d (%c) : ", plays % 2 != 0? 1 : 2,
27             plays % 2 != 0? 'X' : 'O');
28     scanf("%d", &slot);
29     printf("\n");
30
31     row = (slot - 1) / 3;
32     column = (slot - 1) % 3;
33
```

```
34        for(int i = 0; i < 3; i++){
35            printf("      ");
36            for(int j =0; j < 3; j++){
37                if(i == row && j == column)
38                    x[i][j] = plays % 2 != 0? 'X' : 'O';
39
40                printf("%c | ", x[i][j]);
41
42            }
43        printf("\n     -----------\n");
44        }
45
46
47        if(x[0][0] == x[1][1] && x[0][0] == x[2][2] ||
48           x[2][0] == x[1][1] && x[2][0] == x[0][2]){
49           printf("\nplayer %d has Won !!", plays % 2 != 0? 1 : 2);
50           return 0;
51        }
52
53        for(int i = 0; i < 3; i++){
54            if(x[i][0] == x[i][1] && x[i][0] == x[i][2] ||
55               x[0][i] == x[1][i] && x[0][i] == x[2][i]){
56                printf("\nplayer %d has Won !!", plays % 2 != 0? 1 : 2);
57                return 0;
58            }
59        }
60
61        plays++;
62
63        }while(plays <= 9);
64
65        printf("\nGame has ended and no player has won !\n");
66
67        return 0;
```

شرح البرنامج

أول ما نحتاجه هنا هو مصفوفة ثنائية البعد لنعرض عن طريقها الشكل المعهود للعبة، لذلك سنقوم بدايةً بتعريف مصفوفة و إعطائها قيم إبتدائية ليتم إظهارها للاعبين فى بداية اللعبة، كالآتى.

```
char x[3][3] = {{'1','2','3'},
                {'4','5','6'},
                {'7','8','9'}};
```

ثم نقوم بتعريف المتغيرات التى سنحتاجها أثناء كتابة البرنامج.

```
int plays = 1,
    slot,
    row,
    column;
```

المتغير plays سنستخدمه لتخزين عدد اللعبات التى لعبت حتى الآن ، و slot لرقم المربع الذى اختاره اللاعب، row و column لتخزين الصف و العمود للمربع الذى تم إختياره من قبل اللاعب.

ثم نقوم بعرض شكل بداية اللعبة ، عن طريق هذه الحلقات التكرارية المتداخلة.

```
for(int i = 0; i < 3; i++){
     printf("       ");
     for(int j =0; j < 3; j++)
         printf("%c | ", x[i][j]);
printf("\n       -----------\n");
}
```

و أظن أنك تستطيع فهم الكود إذ أننا تعاملنا قبل ذلك كثيراً مع الحلقات التكرارية و فهمنا طريقة عملها، و هذا الكود يقوم بطباعة الشكل الآتى.

```
Welcome to X - O game
---------------------

     1 | 2 | 3 |
     -----------
     4 | 5 | 6 |
     -----------
     7 | 8 | 9 |
     -----------
```

سنقوم الآن بطلب إدخال رقم المربع الذى يختاره اللاعب و حفظه فى المتغير المخصص slot، ثم نقوم بحساب الصف و العمود الخاص بالمربع المختار فى المتغيرين row و column, كالآتى.

```
printf("\nPlayer %d (%c) : ", plays % 2 != 0? 1 : 2,
          plays % 2 != 0? 'X' : 'O');
scanf("%d", &slot);
printf("\n");

row = (slot - 1) / 3;
column = (slot - 1) % 3;
```

فى جملة الطباعة تلاحظ إننا نميز بين اللاعب الأول و اللاعب الثانى عن طريق المتغير plays إذا كان فردياً كان اللاعب الأول هو صاحب اللعبة الحالية، و لو كان زوجياً كان اللاعب الثانى، و نطبع العلامة الخاصة سواء X أو O بكل منهم.أما بالنسبة لتحديد رقم الصف و العمود ، فيمكننا هذا عن طريق طرح واحد من الرقم المدخل إذ أن المصفوفة تبدأ من 0 و ليس من 1 كما يظهر للمستخدم، لذلك فأن أى رقم يدخله المستخدم يكون أكبر بواحد من الحقيقى ، لذا نقوم بطرح هذا الواحد ، ثم نقوم بالقسمة على 3 إذا أردنا الحصول على عدد الصفوف ، و إيجاد باقى القسمة على 3 إذا أردنا الحصول على عدد الأعمدة، و يمكنك تجربتها بنفسك.

الآن نريد أن نحفظ فى المصفوفة العلامة التى أدخلها اللاعب فى المربع الذى إختار قم طباعة الشكل الجديد، و نقوم بهذا عن طريق نفس الحلقات التكرارية التى إستخدمناها فى عرض الشكل الإبتدائى للعبة و لكن مع تغيير بسيط لتظهر العلامة المدخلة بدلاً من الرقم المختار، كالآتى.

```
for(int i = 0; i < 3; i++){
     printf("      ");
     for(int j =0; j < 3; j++){
          if(i == row && j == column)
               x[i][j] = plays % 2 != 0? 'X' : 'O';

          printf("%c | ", x[i][j]);

     }
printf("\n      -----------\n");
}
```

و الآن يجب إختبار عما إذا كان قد فاز أحد اللاعبين أم لا . و نقوم بذلك عن إختار مجموعة من الشروط تقضى بأن الفائز يجب أن يكون قد أكمل صفاً كاملاً أو عموداً كاملاً أو قطراً كاملاً بعلامته الخاصة، و نقوم بهذا الإختبار كالآتى.

```c
if(x[0][0] == x[1][1] && x[0][0] == x[2][2] ||
   x[2][0] == x[1][1] && x[2][0] == x[0][2]){
   printf("\nplayer %d has Won !!", plays % 2 != 0? 1 : 2);
   return 0;
}

for(int i = 0; i < 3; i++){
    if(x[i][0] == x[i][1] && x[i][0] == x[i][2] ||
       x[0][i] == x[1][i] && x[0][i] == x[2][i]){
        printf("\nplayer %d has Won !!", plays % 2 != 0? 1 : 2);
        return 0;
    }
}
```

فى الجملة الشرطية الأولى إختبرنا إكتمال قطر من عدمه، و فى الحلقة التكرارية و الشرط الثانى إختبرنا إكتمال صف أو عمود من عدمه، و فى حال تحقق أى من الشرطين يتم طباعة إسم الفائز ثم إنهاء البرنامج.

هل لاحظت شيئاً ؟ .. هذه الأوامر التى كتبناها يجب أن تتكرر كل مرة فى أدوار اللعب ، فكل مرة يجب علينا أن نطلب إدخال مربع جديد ، ثم نحفظ العلامة الخاصة فى ذلك المربع ، ثم طباعة الشكل الجديد، ثم إختبار عما إذا كان أحد اللاعبان قد فاز. إذاً سنحتاج إلى حلقة تكرارية تضم هذه الأوامر ، و لكن ما الحلقة الأنسب من وجهة نظرك ؟

إجابتك صحيحة ، نعم إنها do-while ، لأننا نقوم بإستخدامها فى حالة أننا نريد تنفيذ مجموعة من الأوامر على الأقل مرة واحدة و هذا واقع حال اللعبة التى نتعامل معها الآن ، و سيكون شرط إستمرار الحلقة هو أن تكون plays أقل من أو يساوى 9 إذ أن أقصى عدد للعبات هو 9 عدد المربعات، لا ننسى أن نزيد العداد – plays – فى كل مرة بمقدار واحد.

و لو إنتهت عدد الأدوار المتاحة و هى 9 و لم يفز أحد وانتهت الحلقة التكرارية do-while يجب طباعة جملة تخطر اللاعبين بأن اللعبة انتهت دون فوز أى منهما، كالآتى.

```c
printf("\nGame has ended and no player has won !\n");
```

التمارين

1) اكتب برنامجاً يستقبل من المستخدم 5 قيم من النوع float يخزنهم فى مصفوفة، ثم يقوم بإنشاء مصفوفة جديدة يخزن فيها عناصر المصفوفة الأولى مرفوعة إلى الأس 5، ثم يطبعها و يطبع مجموع عناصرها كذلك.

2) اكتب برنامجاً يقوم بطباعة جدول مكون عموده الأول من الأرقام من 1 إلى 2 مع زيادة 0,1 فى كل مرة ، و يكون العمود الثانى هو مضاعفات هذه الأرقام، و الثالث هو هذه الأرقام مرفوعة إلى أس 3 ... ، و هكذا حتى أس 5 .

3) اكتب برنامجاً يقوم بطباعة مجموع كل صف على حدة و كذلك كل عمود ، عناصر المصفوفة يقوم بإدخالها المستخدم و كذلك أبعاد المصفوفة.

4) اكتب برنامجاً يقوم بطباعة ما يعرف بالمصفوفة السحرية و هى مصفوفة مربعة ثنائية البعد فيها مجموع أى صف و مجموع أى عمود و مجموع أى قطر متساويين ، و يتم بناء المصفوفة السحرية عن طريق وضع رقم 1 فى منتصف الصف الأول للمصفوفة ثم تنتقل إلى أعلى و يمين و تضع رقم 2 و هكذا حتى تصل إلى آخر عنصر فى المصفوفة التى تساوى قيمته (عدد الصفوف * عدد الأعمدة) ، و إذا حاولت أى قيمة أن تكتب خارج حدود المصفوفة عند حركة (أعلى-يمين) فإنها تنتقل إلى الجانب الآخر من المصفوفة فمثلاً إذا كنت فى الصف الأول و أردت أن تنتقل إلى أعلى فإنك تتجنب الخروج من المصفوفة و تذهب إلى الصف الأخير، و إذا وجد قيمة موجودة من قبل عند محاولة كتابة أى عنصر جديد ، فتكتب هذه القيمة الجديدة تحت القيمة السابقة لها مباشرة و لا تكتب فى مكانها الطبيعى(أعلى-يمين).

الفصل السادس

المتــــغــــيرات الـــنــــصــــيــــة

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ كيفية تعريف المتغيرات النصية فى لغة السى.
- ✓ كيفية إستخدام المتغيرات التصية فى عمليات الإدخال و الإخراج.
- ✓ العمليات المختلفة على المتغيرات النصية .
- ✓ العمليات المختلفة على الأحرف .
- ✓ كيفية إستخدام الدوال الأكثر إستخداماً من string.h و ctype.h .

ذكرنا من قبل أنه لا يوجد متغير من النوع string في لغة السي ، و لكن يمكن عمل متغير مع النوع string عن طريق عمل مصفوفة من النوع char ، و في هذا الفصل سنتناول كيفية التعامل مع الـ strings و العمليات التي يمكننا أن نقوم بها على المتغيرات من النوع String و النوع Char.

تعريف المتغير النصي

نقوم بتعريف المتغير النصي عن طريق تعريف مصفوفة بعدد حروف الـ (String + 1) أو أكبر ، كالتالي .

```
char s[6] = "Hello";
```

كل حرف من هذه الكلمة يسجل كعنصر في المصفوفة ثم يُوضع 0\ بعد آخر حرف وتسمى null character و هذه موجودة للإخطار بأنه قد تم الوصول إلى نهاية الـ string . فيكون شكل كلمة Hello في الذاكرة كالآتي.

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0

و يمكن تعريف أكثر من string في مصفوفة واحدة عن طريق إستخدام المصفوفات ثنائية البعد كالآتي.

```
char s[3][10] = {"Hello", "World","!!"};
```

لاحظ أنه تم هنا تعريف المصفوفة لتحتوي ثلاث متغيرات من النوع String كل واحد منهم يحتوي على 9 أحرف إذا ما راعينا وجود 0\ في نهاية كل كلمة.

عمليات الإدخال و الإخراج مع المتغيرات النصية

يتم إستخدام محدد النوع ٪s كمحدد لطباعة أو إستقبال String .

مثال

```
char s[11];

printf("Enter String ( max 10 char ): ");
scanf("%s", s);
printf("\n%s", s);
```

و خرج البرنامج سيكون كالاتى .

```
Enter String ( max 10 char ):mahmoud
mahmoud
```

إلا أن دالة scanf لا تعمل بشكل جيد مع الـ strings ، لأنك لو أردت مثلاً إدخال جملة كاملة بها مسافات ستتطلب دالة scanf تخزين كل كلمة منها فى متغير لأن وجود مسافة بعد الكلمة يعنى إنتهائها بالنسبة للدالة ، لذلك نستخدم دالة أخرى إسمها gets ، و عن طريقها يمكنك تخزين جملة كاملة فى متغير واحد .

مثال

```
int main(void){

    char s[100];

    printf("Enter more than one string: ");
    gets(s);
    printf("\n%s", s);

    return 0;
}
```

خرج البرنامج سيكون كالتالى.

```
Enter more than one string: this book is Simply C

this book is Simply C
```

إلا أن هناك مشكلة من الممكن أن تحدث عند إستخدام هذه الدالة ، و هى مشكلة حدوث overflow ، أى أن عدد الحروف المدخلة أكبر من عناصر المصفوفة ، ففى المثال السابق على سبيل المثال إذا أدخل المستخدم جملة تجاوز عدد حروفها الـ 100 عنصر سيحدث overflow ، و لغتى الـ C و ++C لا يحتويان على حماية ضد حدوث الـ overflow الذى قد يتسبب فى بعض الأحيان فى الكتابة على بيانات أخرى فى الذاكرة و حدوث مشاكل أخرى جسيمة ، و تترك اللغتان المهمة على النظام المشغل الخاص بك ، فهو من عليه منع أى كتابة تحدث خارج النطاق المحدد ، لذا يتوجب عليك الحذر الشديد عندما يكون نظامك المشغل لا يدعم هذه الخاصية . و لتجنب هذه المشكلة يتم إستخدام دالة fgets ، و يتم إدخال لها 3 متغيرات الأول ، المعامل الأول هو المتغير الذى سيتم إستقبال الـ String فيه ، و الثانى هو الحد الأقصى الذى سيتم إستقباله، و المعامل الثالث هو الجهة التى سيتم إستقبال البيانات منها و فى هذه الحالة هى stdin أى وحدة الإدخال الإعتيادية و فى أغلب الأحيان تكون هى لوحة المفاتيح .

مثال

```
int main(void){

    char s[10];

    printf("Enter string:  ");
    fgets(s, 10 , stdin);
    printf("\n%s", s);

    return 0;
}
```

إذا تم إدخال أى String أكبر من 10 أحرف، سيتم عمل اقتصاص لهذا الـ String و أخذ أول 10 عناصر منه فقط، كالمثال التالى. لاحظ كيف تم اقتصاص النص.

```
Enter string: my name is mahmoud elbarbary

my name i
Process returned 0 (0x0)   execution time : 27.661 s
Press any key to continue.
```

العمليات على المتغيرات النصية

سنستعرض معاً فى هذا الجزء الدوال المستخدمة فى إجراء عمليات على النصوص ، و يتم إستخدام هذه الدوال عن طريق عمل إستيراد للملف string.h .

دالة strlen

تستخدم هذه الدالة فى إيجاد عدد حروف الـ string ، مثال.

```
#include <stdio.h>
#include <string.h>

int main(void){

    char s[11];
    printf("Enter a String: ");
    gets(s);
    printf("%d",strlen(s));

    return 0;
}
```

هنا يتم طباعة عدد أحرف النص الذى تم إدخاله، كالآتى.

```
Enter a String: mahmoud
7
```

كما ترى يتم طباعة عدد أحرف النص و لا يتم إحتساب 0\ التى توضع فى نهاية النص. انتبه: هذه الدالة تحسب عدد أحرف النص و ليس عدد عناصر المصفوفة ،لذلك تم طباعة 7 و ليس 11(طول المصفوفة).

دالتى strcpy و strncpy

هاتان الدالتان يستخدمان فى عمل نسخة من متغير نصى و وضعها فى متغير نصى آخر ،و الفرق بينهما أن strncpy تستطيع بها أن تتحكم فى عدد الحروف التى تريد نسخها و لا تجبرك على نسخ كل الحروف كما فى strcpy. المعامل الأول للدالتين هو المتغير الذى سيتم النسخ إليه ،و المعامل الثانى هو المتغير الذى سيتم

النسخ منه (المصدر) ، و المعامل الثالث موجود فى دالة strncpy فقط و هو يعبر عن عدد الحروف التى سيتم نسخها ، و المثال التالى يوضح طريقة استخدامهما و الفرق بينهما فى الإستخدام.

مثال

```
int main(void){

    char s[11],
         m[11],
         n[11];

    printf("Enter String ( max 10 Char ): ");
    gets(s);
    strcpy(m, s);
    printf("\n%s",m);
    strncpy(n, s, 3);
    n[3] = '\0';
    printf("\n%s", n);

    return 0;
}
```

هنا تم إستقبال نص من المستخدم ثم وضعه فى متغير نصى ، ثم قمنا فى المرة الأولى بعملية نسخ بإستخدام strcpy إلى المتغير m ، و فى المرة الثانية قمنا بعملية النسخ عن طريق strncpy و تم تحديد 3 حروف فقط، و لكن ستلاحظ هنا اننا وضعنا 0\ فى العنصر الرابع للمصفوفة ليتم إنهاء الـنص المكون من الـ 3 أحرف الذى تم نسخهم إليه بشكل صحيح ، لكى لا يحدث مشاكل غير متوقعة عند إستخدام هذا المتغير النصى فى أى عمليات أخرى.

و يكون خرج البرنامج كالتالى

```
Enter String ( max 10 Char ): mahmoud

mahmoud
mah
```

دالتى strcat و strncat

هاتان الدالتان يستخدمان فى وضع نص فى نهاية نص آخر ،و الفرق بينهما أن strncat تستطيع أن تتحكم فى عدد الحروف التى تريد وضعها فى نهاية النص . و المثال التالى يوضح طريقة استخدامهما و الفرق بينهما.

```c
char s[50];

printf("Enter String ( max 10 char ): ");
gets(s);
printf("\n%s", strcat(s, ".Really?"));
printf("\n%s", strncat(s, ".Really?", 5));
```

هنا إستخدمنا دالة strcat فى المرة الأولى، و وضعنا فى نهايتها نص ثابت – و يمكننا أن نضع نص متغير يقوم المستخدم بإدخاله أيضاً . و فى المرة الثانية إستخدمنا strncat و حددنا 5 أحرف فقط.

و مثال على خرج البرنامج .

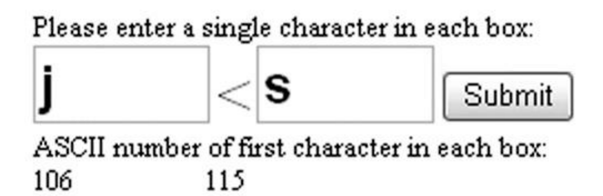
```
Enter String ( max 10 char ): mahmoud

mahmoud.Really?
mahmoud.Really?.Real
```

دالتى strcmp و strncmp

هاتان الدالتان يستخدمان فى المقارنة بين نصين ، و يعودان بـصفر إذا كانا النصين متساويان ، و يعودان بقيمة موجبة إذا كان النص الأول أكبر من النص الثانى، و بقيمة سالبة إذا كان النص الثانى أكبر من النص الأول. و لكن كيف يتم المقارنة بين نصين ؟!

الحاسب الآلى لا يتعامل مع النصوص مباشرة و إنما يتعامل مع الأرقام، ففى الحقيقة أن لكل حرف رقم أو كود يُعبر عنه فيما يعرف بالـ ASCII Code . ففى الصورة الموضحة، تجد أن كود حرف الـ s = 115 و كود حرف الـ j = 106 ، لذلك عند المقارنة بنهم نجد أن الـ s أكبر من الـ j .



فى النصوص يتم مقارنة الحرف الأول بين النصين ، إذا كانوا متساويين يتم الإنتقال إلى الحرفين الثانيين و المقارنة بينهما ، و هكذا حتى يكون أحد الحروف فى أحد النصوص أكبر من الحرف المناظر له فى النص الآخر ، فيكون النص الذى وُجد فيه الحرف الأكبر هو الأكبر . أما إذا تم الانتهاء من جميع الأحرف و جميعهم متساويون ، فيكون النصان متساويين . و انتبه إلى أن s (الأحرف الصغيرة) لا تساوى (الأحرف الكبيرة) S كل منهما له كود مختلف.

مثال

الآن سنتناول مثال يوضح إستخدام الدالتين .

```
int main(void){

    char s[11],
         n[11];

    printf("Enter the first String: ");
    scanf("%s", s);
    printf("Enter the second String: ");
    scanf("%s", n);

    if(strcmp(s, n) == 0)
        printf("\nthe two strings are the same");
    else if(strcmp(s,n) < 0)
        printf("\nthe 1st string < the 2nd string ");
    else
        printf("\nthe 2nd string < the 1st string ");

   if(strncmp(s, n , 3) == 0)
       printf("\nthe first 3 chars of the 2 string are the same");

   return 0;
}
```

فى الجملة الشرطية الأولى تم إستخدام strcpy للمقارنة بين نصين قام بإدخالهم المستخدم، و فى الجملة الشرطية الثانية قمنا بإستخدام strncmp لإختبار إذا كان أول 3 أحرف من النصين متساويان أم لا .

مثال على الخرج

```
Enter the first String: mahmoud
Enter the second String: mahrous

the 1st string < the 2nd string
the first 3 chars of the 2 string are the same
```

هنا كان النص الأول أصغر من الثانى لأنهم تساويا فى أول ثلاث حروف و اختلفا فى الرابع و حرف الـ m أقل من حرف الـ r .

هنا نكون قد تناولنا كيفية إستخدام أشهر دوال العمليات على النصوص إستخداماً، و يمكنك أنت أن تطلع على باقى الدوال و تكتشف إستخدامها بنفسك.

العمليات على الأحرف

من منطلق أن النص ما هو إلا مجموعة من الأحرف ، لذلك قد نريد أن نقوم ببعض العمليات على هذه الأحرف التى هى جزء من النص، و توجد بعض الدوال التى تستخدم فى القيام بهذه العمليات و يتم إستخدامها عن طريق إستخدام الملف ctype.h . و هذه الدوال تنقسم إلى نوعين إحداهما يستخدم فى الإختبار و النوع الآخر يستخدم

للتحويل ، و سنستعرض أهم دوال النوعين مع بعض الأمثلة.

العمل	الدالة
تختبر عما إذا كان الحرف أبجدياً	isalpha()
تختبر عما إذا كان الحرف رقماً	isdigit()
تختبر عما إذا كان الحرف صغيراً	islower()
تختبر عما إذا كان الحرف كبيراً	isupper()
تختبر عما إذا كان الحرف مسافة	isspace()
تحول الحرف إلى صغير	tolower()
تحول الحرف إلى كبير	toupper()

مثال

فى هذا المثال نقوم بطباعة عدد الأحرف الكبيرة فى كلمة يقوم بإدخالها المستخدم .

```
char s[100];
int c = 0;

printf("Enter a word : ");
scanf("%s", s);

for(int i = 0; i < strlen(s); i++){
    if(isupper(s[i])) c++ ;
}

printf("the number of uppercase letters = %d", c);
```

مثال على الخرج

```
Enter a word : MaHmouD
the number of uppercase letters = 3
```

و بالمثل يمكن استخدام islower() إذا أردنا إيجاد عدد الأحرف الصغيرة ، و كذلك مع باقى الدوال الأخرى التى تستخدم لإختبار الأحرف، فيمكنك إيجاد عدد المسافات فى النص الذى تم إدخاله و لكن يجب إستخدام دالة gets فى هذه الحالة و ليس scanf.

مثال

فى هذا المثال نقوم بتحويل أحرف كلمة يقوم بإدخالها المستخدم إلى أحرف كبيرة.

```
char s[100];

printf("Enter a word : ");
scanf("%s", s);

for(int i = 0; i < strlen(s); i++)
    s[i] = toupper(s[i]);

printf("the word after converting it to uppercase : %s", s);
```

مثال على الخرج

```
Enter a word : mAhmOuD
the word after converting it to uppercase : MAHMOUD
```

و بالمثل يمكن إستخدام tolower() إذا أردنا تحويل الأحرف إلى أحرف صغيرة.

برنامج تطبيقي

لاتتطرق إلى دراسة هذا البرنامج التطبيقي إلا بعد الإنتهاء من دراسة فصلى المؤشرات و الدوال

سنقوم بعمل محرر نصوص (text editor) بسيط يقوم بعمليات تعديل على سطر واحد فقط.

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4  #define MAX_LEN 100
5
6  void do_op(char phrase[], char command);
7  void delete_str(char phrase[]);
8  void insert_str(char phrase[]);
9  void find_str(char phrase[]);
10 int get_index(char d[], char phrase[]);
11
12 int main(void){
13
14     char phrase[MAX_LEN];
15     char command;
16
17     printf("Ener the phrase to edit : ");
18     fgets(phrase, MAX_LEN, stdin);
19
20     do{
21         printf("Enter  command ( D: delete , I: insert, F: Find, Q: quit.\n ");
22         scanf(" %c", &command);
23         if(command == 'q')
24             continue;
25         do_op(phrase, command);
26     }while(tolower(command) != 'q');
```

```
27
28         return 0;
29     }
30
31     void do_op(char phrase[], char command){
32
33          switch (tolower(command)){
34                 case 'd':
35                     delete_str(phrase);
36                     printf("\nthe phrase after deletion operation : %s", phrase);
37                     break;
38
39                 case 'i':
40                     insert_str(phrase);
41                     printf("\nthe phrase after insertion operation : %s", phrase);
42                     break;
43
44                  case 'f':
45                      find_str(phrase);
46                      break;
47
48                  default:
49                      printf("the operation is not valid");
50              }
51     }
52
53     void delete_str(char phrase[]){
54
55         char d[MAX_LEN];
56         int index , n;
57
58         printf("\nEnter the substring to be deleted : ");
59         scanf("%s", d);
60
61         n = strlen(d);
62
63         if (index = get_index(d, phrase))
64             if(strlen(phrase) == index + n)
65                 phrase[index] = '\0';
66             else
67                 strcpy(&phrase[index], &phrase[index + n]);
68         else
69             printf("\nnothing found to be deleted");
70
71     }
72
73     void insert_str(char phrase[]){
74
75         char i[MAX_LEN],
76              rest_str[MAX_LEN];
77
```

```
78        int index , n;
79
80        printf("Enter the substring to insert : ");
81        scanf("%s", i);
82
83        printf("Enter the index to insert at : ");
84        scanf("%d", &index);
85
86        n = strlen(i);
87
88        strcpy(rest_str, &phrase[index]);
89        strcpy(&phrase[index], i);
90        strcpy(&phrase[index +n], rest_str);
91
92    }
93
94    void find_str(char phrase[]){
95
96        char i[MAX_LEN];
97        printf("Enter the substring to find: ");
98        scanf("%s", i);
99        if(get_index(i, phrase)){
100           printf("substring at index %d\n", get_index(i, phrase));
101       }else{
102           printf("not found!\n");
103       }
104   }
105
106   int get_index(char d[], char phrase[]){
107
108       int index;
109
110       char *found = strstr(phrase, d);
111
112       if(found != NULL){
113           index = found - phrase;
114           return index;
115       }else
116           return 0;
117   }
118
```

الشرح

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX_LEN 100
```

تم إستيراد كل ملف من هذه الملفات (header files) لأننا سنستخدم منهم دوال خلال هذا البرنامج ، فعلى سبيل التذكرة ، stdio.h لإستخدام الدوال الخاصة بعمليات الإدخال و الإخراج ، و string.h لإستخدام الدوال الخاصة بالعمليات على المتغيرات النصية، و ctype لإستخدام الدوال الخاصة بالعمليات على الحروف. ثم تم تعريف متغير ثابت يُمثل الحد الأقصى لكل المصفوفات التى سنستخدمها فى هذا البرنامج.

```c
void do_op(char phrase[], char command);
void delete_str(char phrase[]);
void insert_str(char phrase[]);
void find_str(char phrase[]);
int get_index(char d[], char phrase[]);
```

لاحظ أن : المصفوفات يمكن إستخدمها كدخل و خرج من الدوال فى نفس الوقت ، لأن اسم الدالة ما هو إلا مؤشر يشير إلى بداية هذه الدالة (لذلك يعود اسم الدالة دائماً بعنوان أول عنصر فى الدالة) ، فهى تقوم بنفس عمل المؤشرات فيما يتعلق بالعمل كخرج و دخل مع الدوال.

هنا تم تعريف الـ prototypes الخاصة بالدوال التى سنستخدمها فى البرنامج :

1) الدالة الأولى do_op تقوم بإستقبال المتغير النصى و نوع الأمر كمعاملين لها ، و فيها سنقوم بتنفيذ العملية المختارة – سنتناول شرح عمل الدالة بالتفصيل لاحقاً - ، و الدالة لا تعود بشىء.

2) الدالة الثانية delete_str تستقبل المتغير النصى ، و فيها سنقوم بإستقبال النص المراد حذفه من هذا المتغير النصى ثم تنفيذ عملية الحذف ، و الدالة لا تعود بشىء.

3) الدالة الثالثة insert_str تستقبل المتغير النصى، و فيها سنقوم بإسقبال النص المراد وضعه فى مكان محدد من هذا المتغير النصى ثم القيام بعملية الإدخال هذه ، و الدالة لا تعود بشىء.

4) الدالة الرابعة find_str تستقبل المتغير النصى، و فيها سنقوم بإستقبال النص المراد البحث عنه فى هذا المتغير ثم طباعة مكانه إن وُجد ، و الدالة لا تعود بشىء.

5) الدالة الخامسة get_index تستقبل متغرين نصيين ، يتم البحث فى النص الثانى عن وجود النص الأول ، و هذه الدالة تعود بمكان النص أو تعود بـ **0** إذا لم يتم إيجاده.

الدالة الرئيسية

```
int main(void){

     char phrase[MAX_LEN];
     char command;

     printf("Ener the phrase to edit : ");
    fgets(phrase, MAX_LEN, stdin);

    do{
        printf("Enter  command ( D: delete , I: insert, F: Find, Q: quit.\n ");
        scanf(" %c", &command);
        if(command == 'q')
            continue;
        do_op(phrase, command);
    }while(tolower(command) != 'q');

    return 0;
}
```

قمنا بتعريف متغير نصى إسمه phrase لنستقبل فيه النص الذى يريد المستخدم تعديله ، و كذلك command لإستقبال الحرف الذى يعبر عن العملية المراد تنفيذها . ثم قمنا بإستقبال النص المراد تعديله عن طريق fgets.

الحلقة التكرارية سيتم تنفيذها على الأقل مرة لذا إستخدمنا الحلقة التكرارية do-while . نقوم فى كل مرة بإستقبال الأمر المراد تنفيذه ، ثم إختبار عما إذا كان هذا الأمر q أى يعنى إغلاق البرنامج فحينها نقوم بعمل continue لأننا لا نحتاج لتفيذ do_op فى هذه الحالة ، و حينها سيتم إنهاء البرنامج. أما فى حالة أى اختيار آخر سيتم تنفي الدالة do-op .

دالة do_op

```
void do_op(char phrase[], char command){

     switch (tolower(command)){
            case 'd':
                delete_str(phrase);
                printf("\nthe phrase after deletion operation : %s", phrase);
                break;

            case 'i':
                insert_str(phrase);
                printf("\nthe phrase after insertion operation : %s", phrase);
                break;

            case 'f':
                find_str(phrase);
                break;

            default:
                printf("the operation is not valid");
        }
}
```

الدالة do_op عبارة فقط عن switch case تقوم بإستدعاء الدالة المناسبة للأمر المراد تنفيذه . لا يوجد مشكلة فى فهم كيفية عملها ، فسنقوم الآن بإستعراض عمل باقى الدوال.

دالة delete_str

```
void delete_str(char phrase[]){

    char d[MAX_LEN];
    int index , n;

    printf("\nEnter the substring to be deleted : ");
    scanf("%s", d);
```

```
    n = strlen(d);

    if (index = get_index(d, phrase))
        if(strlen(phrase) == index + n)
            phrase[index] = '\0';
        else
            strcpy(&phrase[index], &phrase[index + n]);
    else
        printf("\nnothing found to be deleted");

}
```

فى بداية الدالة قمنا بتعريف متغير نصى d لإستقبال النص المراد حذفه من النص الأصلى، و متغيرين index لتخزين بداية النص المراد حذفه فى النص الأصلى ، و n لتخزين طول المتغير المراد حذفه.

نقوم بإستقبال النص المراد حذفه، ثم يتم تخزين طوله فى n .

الجملة الشرطية الخارجية تقوم بإستخدام دالة get_index لإيجاد مكان النص المراد حذفه فى النص الأصلى و تخزين قيمة الرجوع فى index – لاحظ أن عملية التخزين تمت فى جملة الشرط -، أو يقوم بطباعة إخطار بعدم وجود هذا النص فى النص الأصلى – و فى هذه الحالة تعود الدالة get-index بـ **0** .

فى حالة عودة get_index بقيمة، يتم تنفيذ الجملة الشرطية الداخلية : فى حالة أن النص المراد حذفه هو آخر جزء من النص الأصلى - يختبره هذا الشرط (strlen(phrase) = index + n) - سنقوم بوضع **0** \ عند بداية النص المراد حذفه، ليتم تجاهل (حذف) هذا النص . أما فى حالة أن النص المراد حذفه ليس هو آخر جزء من النص الأصلى ، فيتم نسخ ما بعد النص المراد حذفه على النص المراد حذفه ، فبالتالى يتم حذف النص المراد حذفه من وسط النص، و ذلك عن طريق الأمر التالى.

```
strcpy(&phrase[index], &phrase[index + n]);
```

تم إستخدام & ، لأننى أريد العنصر الموجود فى هذا العنوان و ما بعده حتى النهاية (جزء النص الموجود بعد الجزء المراد حذفه) ، أما إذا لم نستخدم & ستأتى بعنصر واحد فقط رتبته (index + n) .

دالة get_index

```
int get_index(char d[], char phrase[]){

    int index;

    char *found = strstr(phrase, d);

    if(found != NULL){
        index = found - phrase;
        return index;
    }else
        return 0;
}
```

مهمة هذه الدالة أنها تأتى بمكان (رتبة أول عنصر فيه) الجزء المراد حذفه .

نقوم بتعريف متغير index لتخزين موضع هذا الجزء من النص الأصلى ، و نستخدم دالة strstr و هى دالة موجودة فى string.h ، و هذه الدالة تقوم بالبحث عن نص معين فى نص آخر ، المعامل الأول لها هو النص الذى سيتم البحث فيه (phrase فى هذا المثال) ، و المعامل الثانى هو الجزء الذى سنبحث عنه (d) ، و هذه الدالة تعود بعنوان هذا الجزء فى الذاكرة أو تعود بـ Null فى حالة عدم وجوده . انتبه : عنوان هذا الجزء فى الذاكرة و ليس مكانه فى النص الذى يتم البحث فيه (نقوم فى هذا المثال بتخزين هذا العنوان فى مؤشر من النوع char اسمه found*).

إذا كان هذا الجزء غير موجود فى النص الأصلى ستعود الدالة بـ 0 ، أما فى حالة وجود فيتم حساب مكان هذا الجزء فى النص الأصلى و العودة به عن طريق هذه الأمر

```
index = found - phrase;
```

و هو يقوم بإيجاد الفارق بين عنوان بداية هذا الجزء و بداية النص الأصلى ، فيحصل على رتبة أول عنصر فى هذا الجزء (بداية هذا الجزء) . تذكر : عند إستخدام اسم المصفوفة فقط دون [] فهى تعود بعنوان أول عنصر فيها (أى عنوان بدايتها) .

دالة insert-str

```
void insert_str(char phrase[]){

    char i[MAX_LEN],
         rest_str[MAX_LEN];

    int index , n;

    printf("Enter the substring to insert : ");
    scanf("%s", i);

    printf("Enter the index to insert at : ");
    scanf("%d", &index);

    n = strlen(i);

    strcpy(rest_str, &phrase[index]);
    strcpy(&phrase[index], i);
    strcpy(&phrase[index +n], rest_str);

}
```

عمل هذه الدالة هو إدخال نص فى مكان معين من النص الأصلى، لذلك تم تعريف متغير نصى i لإستقبال النص المراد إدخاله من المستخدم فيه، index لتخزين المكان المراد وضع هذا النص فيه . و تستخدم n لحفظ طول النص المُدخل .

عملية الإدخال تتم كالتالى :

1) يتم نسخ النص من بداية الموضع المراد إدخال النص المُدخل فيه حتى نهاية النص الأصلى إلى متغيرنصى وسيط يسمى rest_str .

```
strcpy(rest_str, &phrase[index]);
```

2) يتم نسخ النص المُدخل i إلى المكان المراد وضعف فيها .

```
strcpy(&phrase[index], i);
```

3) يتم نسخ ما فى المتغير rest_str إلى النص الأصلى و لكن بعد النص المُدخل مباشرة.

```
strcpy(&phrase[index +n], rest_str);
```

دالة find_str

```
void find_str(char phrase[]){

    char i[MAX_LEN];
    printf("Enter the substring to find: ");
    scanf("%s", i);
    if(get_index(i, phrase)){
        printf("substring at index %d\n", get_index(i, phrase));
    }else{
        printf("not found!\n");
    }
}
```

الدالة تقوم بالبحث عن نص معين فى النص الأصلى، لذا يتم إستقبال النص المراد البحث عنه فى المتغير النصى i ، ثم يتم طباعة مكانه بإستخدام الدالة get_index فى حالة رجوعها بقيمة ، أما فى حالة عدم رجوعها بقيمة فيتم إخطار المستخدم بأن النص محل البحث غير موجود.

إختبار البرنامج

حذف جزء معين من جملة :

```
Ener the phrase to edit : mahmoud ahmed hassan
Enter  command ( D: delete , I: insert, F: Find, Q: quit.
 d

Enter the substring to be deleted : ahmed

the phrase after deletion operation : mahmoud  hassan
```

إضافة جزء معين إلى الجملة :

```
Enter  command ( D: delete , I: insert, F: Find, Q: quit.
 i
Enter the substring to insert : mohammed
Enter the index to insert at : 8

the phrase after insertion operation : mahmoud mohammed hassan
```

إيجاد كلمة معينة فى الجملة :

```
Enter  command ( D: delete , I: insert, F: Find, Q: quit.
 f
Enter the substring to find: ss
substring at index 19
```

إغلاق البرنامج :

```
Enter  command ( D: delete , I: insert, F: Find, Q: quit.
 q

Process returned 0 (0x0)   execution time : 63.849 s
Press any key to continue.
```

التمارين

1) اكتب برنامجاً يقوم بإستقبال الإسم الثلاثى للمستخدم بحيث يفصل بين كل اسم و الآخر " . " نقطة ، ثم يقوم البرنامج بالتخلص من هذه النقط و طباعة الإسم كاملاً مفصولاً بين كل اسم و الآخر بمسافة tab ، كالمثال الآتى .

إذا أدخل المستخدم الإسم الآتى :

Ahmed . Ali . Mokhtar

فيقوم البرنامج بطباعته كالآتى :

Ahmed   Ali   Mokhtar

2) اكتب برنامجاً يقوم بتحويل أى عدد رقمى مكون على الأكثر من 3 أرقام إلى كتابى – العدد يقوم بإدخاله المستخدم – فمثلاً إذا قم المستخدم بإدخال 122 يقوم البرنامج بطباعة "One Hundred twenty two " .

3) اكتب برنامجاً يقوم بإختبار جملة يقوم بإدخالها المستخدم عما إذا كانت Palindrome - أى إذا تم عكس الجملة تعطى نفس الجملة الأصلية – أم لا ، الجملة الآتية مثال على الجملة الـ palindrome .

Never a foot too far, even

4) اكتب برنامجاً يقوم بتحويل HexaDecimal إلى Octal ، و العكس .

الفصل السابع

المؤشرات و حجز الذاكرة ديناميكياً

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ ما هى المؤشرات ، و كيفية تعريفها .
- ✓ الإستخدامات المختلفة للمؤشرات.
- ✓ لماذا نلجأ إلى حجز الذاكرة ديناميكياً .
- ✓ الدوال المستخدمة فى عملية حجز الذاكرة ديناميكياً.

يُفضل دائماً تخيل الذاكرة دائماً كمصفوفة كبيرة جداً جداً من الـ bytes ، لأن هذا سيُسهل عليك كثيراً فهم المؤشرات و كيفية عملها .

ما هى المؤشرات (pointers)

هى متغيرات تحتوى بداخلها على عنوان متغير آخر ، و نوع المؤشرات يكون مثل نوع المتغير الذى يحمل المؤشر عنوانه (أو بمعنى آخر مثل نوع المتغير الذى يشير إليه).

كيفية تعريف المؤشرات

و يتم تعريف المؤشر بنفس طريقة تعريف أى متغير عادى ، و لكن يتم إستخدام * (asterisk) قبل إسم المتغير

مثال

```
int main(void){

    int x;

    int *p;

    return 0;
}
```

فى هذا البرنامج ، قمنا بتعريف متغير x من النوع int ، و قمنا بتعريف مؤشر p يشير إلى متغير من النوع int (لم يتم إعطاؤه قيمة إبتدائية حتى الآن). دعنا نلقى نظرة على الذاكرة فى هذه الحالة.

العنوان	الذاكرة RAM	اسم المتغير
0x1000		x
0x1001		p

كلا المتغيرين لم يتم وضع قيم فيهم حتى الآن . نقوم الآن بإعطائهم قيم إبتدائية ، كالآتى.

```
int main(void){

    int x = 20;

    int *p = &x;

    return 0;
}
```

لاحظ كيف تم إعطاء قيمة إبتدائية للمؤشر ، إستخدمنا & و تستخدم للإتيان بعنوان المتغير x (تطلب من البرنامج أن يأتى لها بعنوان x) ، ثم يتم وضع عنوان x بداخل المؤشر.

الذاكرة فى هذه الحالة ستكون على هذا الشكل.

العنوان	الذاكرة RAM	اسم المتغير
0x1000	20	x
0x1001	1000	p

لاحظ أنه تم وضع عنوان x (1000) بالهكسا فى المتغير p .

و الآن سنقوم بطباعة قيمة x مرتين ، مرة بطريقة مباشرة عن طريق طباعة x ، و مرة أخرى بطريقة غير مباشرة عن طريق طباعة القيمة التى توجد بالعنوان الذى يحمله المؤشر.

```
int main(void){

    int x = 20;

    int *p = &x;

    printf("%d\n", x);
    printf("%d\n", *p);

    return 0;
}
```

لاحظ هنا كيف تم الإتيان بقيمة x عن طريق المؤشر p الذى يحمل عنوانه ، عن طريق إستخدام * قبل اسم المؤشر .

سيكون ناتج جملتى الطباعة متساوى كما توقعنا .

```
20
20
```

حتى الآن قابلنا إستخدامين لـ * مع المؤشرات :

1) عندما تستخدم أثناء تعريف المؤشر .
2) عندما تستخدم مع المؤشر فى أى موضع آخر – غير التعريف – و تقوم فى هذه الحالة بالإتيان بالقيمة التى توجد بالعنوان الذى يحمله المؤشر.

الآن نقوم بطباعة عنوان x بطريقتين مختلفتين ، مرة بإستخدام & مع المتغير ، و مرة أخرى بطباعة p .

```
int main(void){

    int x = 20;

    int *p = &x;

    printf("%p\n", &x);
    printf("%p\n", p);

    return 0;
}
```

لاحظ إستخدام p٪ عند طباعة عنوان . عند تنفيذ هذا البرنامج ، ستجد أن كلا الجملتين يقوموا بطباعة نفس العنوان.

نعلم أن لأى متغير عنوان فى الذاكرة .. هل المؤشرات لها عناوين ؟ ... بالطبع ، لأن المؤشرات مثلها مثل أى متغير تحجز مكان بالذاكرة لتخزين البيانات فيه ، و أى مكان بالذاكرة لابد و أن يكون له عنوان خاص به ، و المثال السابق كان عنوان المؤشر p هو 1001 كما نرى .

العنوان	الذاكرة RAM	اسم المتغير
0x1000	20	x
0x1001	1000	p

يوجد للمؤشرات إستخدامات عديدة ، مثل :

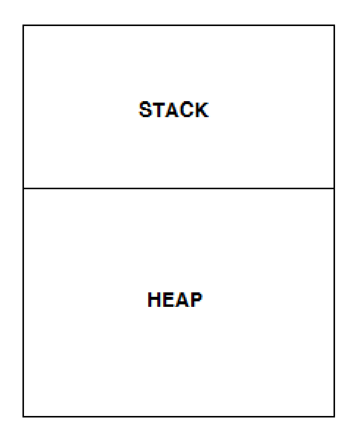
1) حجز الذاكرة ديناميكياً (Dynamic memory allocation) .
2) إستخدامها مع الدوال ، للرجوع بأكثر من خرج من الدالة ، أو إستخدام المؤشر الواحد كدخل و خرج لهذه الدالة فى آن واحد – سيتم دراسته فى فصل الدوال .
3) إستخدامها عند التعامل مع الملفات – سيتم دراسته فى فصل الملفات.

لذلك فإنه من الأهمية بمكان إستيعابك الجيد للفكرة العامة للمؤشر ، و كيفية عمله ، لما سيبنى عليه من مواضيع متقدمة عديدة ، سنتناولها بإذن اللّه فى هذا الكتاب.

Stack / Heap

يعتمد فهمك لعملية حجز الذاكرة ديناميكياً على درايتك بكيفية عمل الذاكرة و كيفية تقسيمها و كيف يتم إستخدامها ، لذلك نبدأ بالتطريق إلى نقطة هامة و هى الفرق بين الـ stack و الـ heap .

كلاهما جزء من الذاكرة المؤقتة RAM . كما هو موضح بالشكل الآتى . و لكننا فى الأشكال التوضيحية سنرسمهم منفصلين فقط للتبسيط .



يتم تخزين قيمة المتغيرات المحلية فى الـ heap ، بينما يتم تخزين أسماء هذه المتغيرات (refrences) فى الـ stack .

مثال

هذا البرنامج يقوم بتعريف مجموعة مختلفة من المتغيرات ، و إعطائهم قيم إبتدائية .

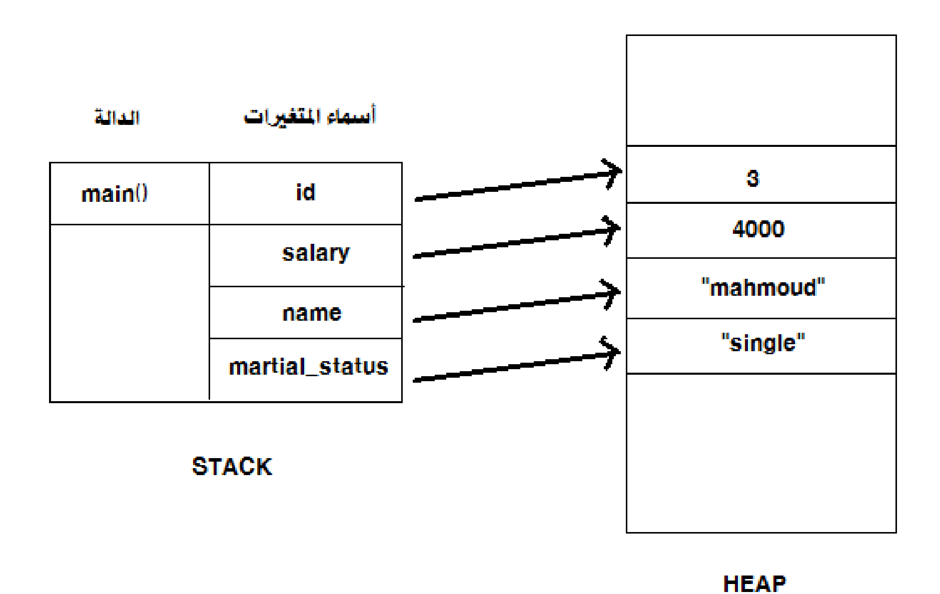
```
int main(void){

    int id = 3,
        salary = 4000;

    char name[] = "Mahmoud",
         martial_status[] = "Single";

    return 0;
}
```

سيتم تخزين هذه المتغيرات فى الذاكرة كالآتى .



الفرق بين الـ Stack و الـ Heap

الـ stack تقوم بحذف البيانات المخزنة فيها تلقائياً عند الإنتهاء من تنفيذ الدالة ، فى حين أن الـ heap يجب عليك أنت من يقوم بعملية الحذف – و سيتضح لنا كيف نقوم بهذا لاحقاً فى هذا الفصل.

الـ stack تعمل بنظام LIFO إختصاراً لـ Last Input First Output ، أى أن آخر جزء تم حجزه من الذاكرة هو أول جزء سيتم حذفه إذا تمت عملية تفريغ للـ stack ، و هذه العملية تلقائية لا تتطلب تدخلاً من المبرمج. فى حين أن الـ Heap لا يقيدك بهذا النظام ، و يعطيك كامل الحرية فى حجز أى جزء من الذاكرة أو حذف أى جزء من الذاكرة فى أى وقت ، لذلك كانت عملية الحجز فيها تتصف بالديناميكية نسبة إلى الحرية فى التعامل معها .

الـ stack بسيط لذلك هو الأسرع فى عملية التخزين ، و كذلك محدود المساحة مقارنة بالـ heap .

متى نستخدم عملية حجز الذاكرة ديناميكياً ؟

يُفضل دائماً الإبتعاد عن إستخدام هذه العملية لأنها أكثر صعوبة فى إستعمالها و كذلك لأنها تتحكم فى الذاكرة بشكل مباشر مما قد يسبب أخطاء كارثية .

هناك حالات معينة عليك فيها أن تقوم بحجز الذاكرة ديناميكياً :

1) عدم علمك مُسبقاً بحجم هيكل بيانى مُعين و ليكن " مصفوفة " .
2) عندما تريد إستخدام مساحة كبيرة من الذاكرة ، لأن مساحة الـ stack كما قلنا محدودة فعند تخزين فيها بيانات أكبر من المساحة المتاحة ، يتسبب ذلك فى حدوث ما يسمى بـ stack overflow ، و حدوث crash للبرنامج .
3) عندما تريد إبقاء محتويات الدالة كما هى بعد إنتهاء إستدعائها، و لما كانت الـ stack تقوم بعملية حذف تلقائية عند انتهاء تنفيذ الدالة ، وجب حجز الذاكرة ديناميكياً أى إستخدام الـ heap .

Dynamic Memory Allocation

بعد أن تعرفنا على مفهوم الـ stack و الـ heap ، و متى بالتحديد يتم حجز الذاكرة ديناميكياً ، سنقوم بشرح كيفية حجز الذاكرة ديناميكياً عن طريق المؤشرات (pointers) .

مثال

هذا البرنامج يقوم بتعريف متغير نصى " مصفوفة" ثم يستخدم مؤشر يحمل عنوان أول عنصر بهذه المصفوفة.

```
int main(void){
    char s[] = "ME";
    char *p = s;
    return 0;
}
```

لاحظ أنه عند إستخدام اسم المصفوفة فقط تعود لنا بعنوان أول عنصر فيها ، لذلك لم نستخدم & .

لنلقى نظرة على شكل الذاكرة فى هذه الحالة.

العنوان	RAM الذاكرة	اسم المتغير
0x1000	'M'	s[0]
0x1001	'E'	s[1]
0x1002	\0	s[2]
0x1003	1000	p

فى هذه الحالة سيحمل المؤشر عنوان اول عنصر فى المصفوفة = 1000 ، و الآن نقوم بتعديل بسيط على البرنامج .

```
int main(void){
    char s[] = "ME";
    char *p = s;
    p++;
    return 0;
}
```

قمنا فقط بزيادة قيمة المؤشر بواحد ، الآن سيكون شكل الذاكرة بعد تنفيذ البرنامج كالآتى .

العنوان	الذاكرة RAM	اسم المتغير
0x1000	'M'	s[0]
0x1001	'E'	s[1]
0x1002	\0	s[2]
0x1003	1001	p

تم زيادة القيمة التى يحملها المؤشر بواحد ، أى أصبحت 1001 بدلاً من 1000 ، أى أن المؤشر الآن صار يشير إلى العنصر الثانى من المصفوفة ، و فى كل مرة سيتم زيادة قيمة المؤشر بواحد سينتقل إلى العنصر الذى يليه .

الآن سأتطرق إلى شرح الهيكل البيانى structure ، لأنه من أهم المواضيع التى تستخدم فيها Dynamic memory allocation ، ثم نتناول شرح الدوال المستخدمة فى عملية حجز الذاكرة ديناميكياً.

structure

تعرفنا سابقاً على أنواع كثيرة للمتغيرات مثل int ، char و غيرهم ، و لكن هل يمكن عمل متغير من نوع إسمه "إنسان" مثلاً ؟ .. يحمل جميع خصائص الإنسان مثل الإسم و العمر و الحالة الإجتماعية ... إلخ ؟ .. نعم ، و سنقوم نحن بصناعة هذا النوع و تحديد خصائصه عن طريق الهياكل البيانية structures .

فيمكننا مثلا تعريف نوع جديد إسمه human ، و تحديد خصائصه ، كالتالى .

```
typedef struct{

    char name[20];
    int age;
    char martial_status[10];

}human;
```

هذا نوع متغير جديد إسمه " human " ، و خصائصه عبارة عن متغيرات من أنواع مختلفة ، على عكس المصفوفات التى يجب أن يكون كل عناصرها من نفس النوع . و يمكننا تعريف نوع المتغير الجديد بأكثر من طريقة ، و لكن سنكتفى هنا بعرض هذه الطريقة.

يمكننا تعريف متغير من هذا النوع الجديد كالآتى.

```
human mahmoud;
```

عملية تعريف متغير جديد من هيكل معين مشابهة تماماً لنفس عملية تعريف متغير عادى. نقوم بكتابة نوع المتغير أولاً ثم إسم المتغير .

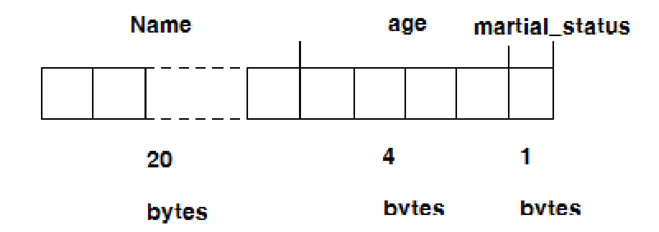
و هذا المتغير يمكننا إعطاء قيم إبتدائية له مثله أيضاً مثل المتغير العادى ، كالآتى .

```
strcpy(mahmoud.name, "mahmoud elbarbary");
mahmoud.age = 21;
mahmoud.martial_status = 's';
```

لاحظ أنه يتم التحكم فى خصائص متغير هيكلى معين عن طريق إسمه ثم نقطة ثم إسم المتغير الداخلى ، و لاحظ أننا إستخدمنا الدالة strcpy لإعطاء قيمة إبتدائية للمتغير name ، و لا يتم إعطاؤه قيمة مباشرة عن طريق assignment كغيره من المتغيرات ، لأنه متغير من النوع String .

و يمكننا تعريف عدد غير محدود من المتغيرات من النوع الهيكلى human ، كل متغير له إسم مميز له و خصائص لها قيم مختلفة عن الآخر.

شكل المتغير الهيكلى mahmoud فى الذاكرة سيكون كالآتى .



و توضع القيم الإبتدائية التى أعطيناها للمتغيرات فى المكان المحجوز لها.

المؤشرات مع الهياكل البيانية

يمكننا إستخدام المؤشرات مع الهياكل البيانية كغيرها من أنواع المتغيرات الأخرى ، فمثلاً إذا أردنا أن نقوم بتعريف مؤشر من النوع human (أى أنه مؤشر يشير إلى متغير من النوع human) ، سنقوم بالطريقة الطبيعية لتعريف أى مؤشر من أى نوع (نوع المتغير ثم مسافة ثم * ثم اسم المؤشر)، كالآتى.

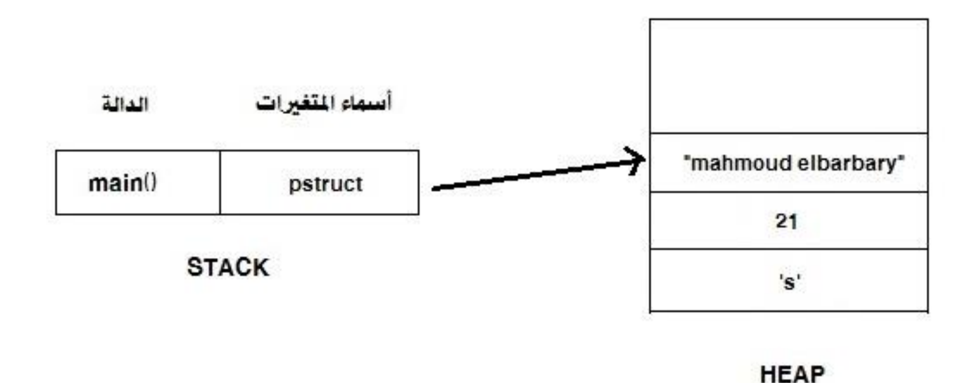
```
human *pstruct ;
```

فى الأمر السابق ، تم تعريف مؤشر اسمه pstruct من النوع human .

يمكننا إعطاء هذا المؤشر قيمة إبتدائية (سيحمل المؤشر متغير من النوع human ، و ليكن mahmoud) ، و نقوم بذلك كالآتى .

```
pstruct = &mahmoud;
```

الآن أصبح المؤشر pstruct يحمل عنوان المتغير mahmoud ، و سيكون شكله فى الذاكرة فى هذه الحالة كالآتى.



تعلمنا من قبل كيفية تغيير قيمة متغير معين عن طريق المؤشر الذى يحمل عنوانه ، و كان ذلك بإستخدام * ، وكذلك يمكننا تغيير القيم الداخلية للمتغير الهيكلى عن طريق * كالآتى .

```
(*pstruct).age = 22;
```

هنا قمنا بتغير قيمة المتغير الداخلى age من المتغير الهيكلى mahmoud الذى يشير عليه المؤشر pstruct ، و لكن يتم استخدام تعبير بديل مساوى لهذا الأمر ، و يكون بهذا الشكل .

```
pstruct -> age = 22;
```

و هو مساوى تماماً للتعبير السابق ، و هو التعبير الأكثر إستخداماً ، لذلك دائماً ما سنستخدمه عند تغيير قيم متغير هيكلى معين بطريقة غير مباشرة عن طريق مؤشر .

دوال حجز الذاكرة

فى عملية حجز الذاكرة ديناميكياً ، تُستخدم دالتين هم calloc و malloc ، كلتا الدالتين يوجدان فى stdlib.h ، لذلك يجب عليك عمل include لهذا الملف قبل إستخدام أياً من الدالتين ، الآن نبدأ بشرح malloc .

دالة malloc

تستخدم هذه الدالة فى حجز مكان لمتغير واحد فى الذاكرة سواء كان هذا المتغير built-in مثل int و char .. إلخ ، أو متغير user-defined مثل المتغير الهيكلى mahmoud من النوع human الذى قمنا نحن بإنشائه .

هذه الدالة تستقبل معامل واحد و هو المساحة التى يشغلها المتغير الذى سيتم حجز هذا الجزء من الذاكرة له ، و نقوم بذلك عن طريق sizeof ، التى تعود بالمساحة التى يشغلها نوع متغير معين .

و دالة malloc تعود بمؤشر يشير إلى عنوان الجزء الذى تم حجزه فى الذاكرة و هذا المؤشر من النوع * void أى انه لا نوع له ، و لكى يتم إستقبال فى مؤشر من نوع آخر لا بد من عمل casting لنوع المؤشر الذى سيتم حفظه فيه .

مثال

فمثلاً إذا أردت أن أقوم بحجز جزء فى الذاكرة لمتغير من النوع int ، سنقوم بالآتى .

```
int *pnum;

pnum = (int *)malloc(sizeof(int));

*pnum = 25;
```

لاحظ هنا أننا قمنا بتعريف مؤشر من النوع int لإستقبال عنوان المكان الذى تم حجزه بواسطة malloc ، مساحة هذا المكان هو الحجم الذى يشغله أى متغير من النوع int غالباً 4 بايت ، و لا تنسى عملية الـ casting ، ثم تم وضع قيمة 25 فى هذا المكان المحجوز عن طريق المؤشر بإستخدام * .

و بالمثل يمكننا حجز مساحة نوع متغير human بنفس الطريقة ، كالآتى .

```
human *phuman;

phuman = (human *)malloc(sizeof(human));

strcpy(phuman -> name , "ahmed");
phuman -> age = 13;
phuman -> martial_status = 'm';
```

تم تغيير قيم المتغيرات عن طريق المؤشر كما تعلمنا سابقاً عن طريق <- .

دالة calloc

تستخدم دالة calloc فى حجز جزء من الذاكرة على شكل مصفوفة ، تستقبل هذه الدالة معاملين ، المعامل الأول هو عدد عناصر هذه المصفوفة، و المعامل الثانى هو حجم العنصر الواحد .

مثال

استقبال عدد معين من الأرقام من المستخدم، ثم تخزين الأرقام فى الذاكرة بإستخدام calloc على شكل مصفوفة .

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int *pnum;
    int n;

    printf("Enter the number of numbers: ");
    scanf("%d", &n);

    pnum = (int *)calloc(n, sizeof(int));

    printf("Enter the numbers : ");
```

```
    for(int i = 0; i < n; i++)
        scanf("%d", &pnum[i]);

    printf("the numbers you entered are : \n");

    for(int i = 0; i < n; i++)
        printf("%d\t", pnum[i]);

    return 0;
}
```

شرح المثال

```
pnum = (int *)calloc(n, sizeof(int));
```

لاحظ أن دالة calloc مشابهة لدالة malloc، إلا أنها تستقبل عدد عناصر المصفوفة .

```
for(int i = 0; i < n; i++)
    scanf("%d", &pnum[i]);
```

هذه الحلقة التكرارية تقوم بإستقبال العناصر كلها ، و حفظها فى pnum[i] ، لاحظ كيف أصبحنا نستخدم المؤشر كالمصفوفة تماماً.

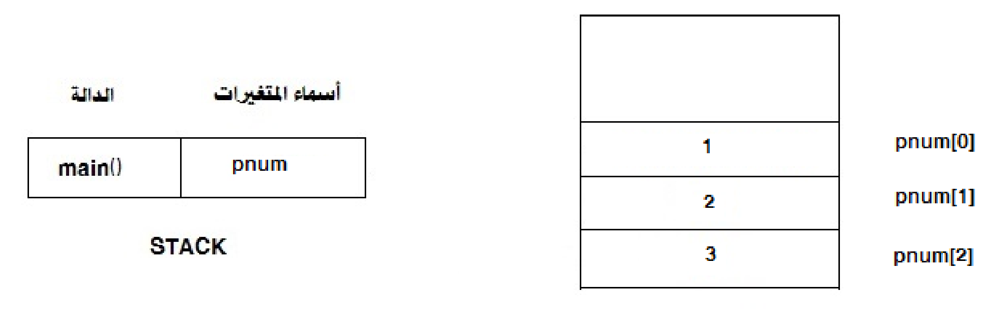
```
for(int i = 0; i < n; i++)
    printf("%d\t", pnum[i]);
```

هنا تم طباعة العناصر كلها بنفس الطريقة المتبعة عند طباعة مصفوفة عادية.

خرج المثال

```
Enter the number of numbers: 3
Enter the numbers : 1 2 3
the numbers you entered are :
1       2       3
```

شكل الذاكرة



مثال

استقبال متغير نصى من المستخدم و حفظه فى الذاكرة فى شكل مصفوفة من الحروف ، ثم طباعته مرة أخرى.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void){

    char *pstring;
    int n;

    printf("Enter the length of the string : ");
    scanf("%d", &n);

    pstring = (char *)calloc(n, sizeof(char));

    printf("Enter the string : ");
    scanf("%s", pstring);

    for(int i = 0; i < n; i++)
        printf("%c", pstring[i]);

    return 0;
}
```

بنفس الطريقة تم عمل مصفوفة من الحروف (متغير نصى) بإستخدام calloc .

خرج المثال

```
Enter the length of the string : 10
Enter the string : Pearce
Pearce
```

شكل الذاكرة

الدالة	أسماء المتغيرات
main()	pstring

STACK

'P'	pstring[0]
'e'	pstring[1]
'a'	pstring[2]
'r'	pstring[3]
'c'	pstring[4]
'e'	pstring[5]
'\0'	pstring[6]

HEAP

مثال

برنامج يقوم بإستقبال بيانات مجموعة من الموظفين ، ثم حفظها فى الذاكرة بإستخدام calloc ، و يقوم بطباعة بيانات جميع الموظفين عند الإنتهاء من عملية إدخال البيانات.

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int n;

    typedef struct{
        int id;
        char name[20];
        int salary;
    }employee;


    employee *pemp;
    printf("Enter the number of employees : ");
    scanf("%d", &n);

    pemp = (employee *)calloc(n, sizeof(employee));

    for(int i = 0; i < n; i++){
        printf("Enter Employee No. %d data : ", i);
        scanf("%d%s%d", &pemp[i].id, pemp[i].name, &pemp[i].salary);
    }

    printf("Retrieving All employees' data\n");

    for(int i = 0; i < n; i++){
        printf("\nEmployee No. %d data\n", i);
        printf("name : %s\n", pemp[i].name);
        printf("id : %d\n", pemp[i].id);
        printf("salary : %d\n", pemp[i].salary);
    }

    return 0;
}
```

شرح المثال

```
typedef struct{
    int id;
    char name[20];
    int salary;
}employee;
```

هنا قمنا بتعريف نوع متغير جديد اسمه employee ، و بداخله قمنا بتعريف خصائص (المتغيرات الداخلية) لهذا النوع الجديد .

```
employee *pemp;
```

هنا قمنا بتعريف مؤشر اسمه pemp من النوع الجديد الذى قمنا بتصميمه employee .

```
pemp = (employee *)calloc(n, sizeof(employee));
```

هنا قمنا بعمل مصفوفة من المتغيرات من النوع employee عدد عناصرها n .

```
for(int i = 0; i < n; i++){
    printf("Enter Employee No. %d data : ", i);
    scanf("%d%s%d", &pemp[i].id, pemp[i].name, &pemp[i].salary);
}
```

هذه الحلقة التكرارية تقوم بإستقبال بيانات كل الموظفين ، لاحظ كيفية تغيير بيانات كل موظف على حدة .

```
for(int i = 0; i < n; i++){
    printf("\nEmployee No. %d data\n", i);
    printf("name : %s\n", pemp[i].name);
    printf("id : %d\n", pemp[i].id);
    printf("salary : %d\n", pemp[i].salary);
}
```

هذه الحلقة التكرارية تقوم بالمرور على بيانات كل موظف و طباعتها.

خرج البرنامج

```
Enter the number of employees : 3
Enter Employee No. 0 data : 1 mahmoud 3000
Enter Employee No. 1 data : 2 khalil 4000
Enter Employee No. 2 data : 3 reda 5000

Retrieving All employees' data

Employee No. 0 data
name : mahmoud
id : 1
salary : 3000

Employee No. 1 data
name : khalil
id : 2
salary : 4000

Employee No. 2 data
name : reda
id : 3
salary : 5000
```

شكل الذاكرة

الدالة	أسماء المتغيرات
main()	pemp

STACK

Value	Variable
1	pemp[0].id
"mahmoud"	pemp[0].name
3000	pemp[0].salary
2	pemp[1].id
"khalil"	pemp[1].name
4000	pemp[1].salary
3	pemp[2].id
"reda"	pemp[2].name
5000	pemp[2].salary

HEAP

دالة free

تقوم دالة free بإعادة الجزء الذى تم حجزه إلى الذاكرة مرة أخرى ، ليمكننا القيام بإستخدامه فى عمليات أخرى ، و يُفضل أن يتم إعادة أى جزء تم حجزه من الذاكرة إلى الذاكرة مرة أخرى عند الإنتهاء من إستخدامه. ففى مثال المتغير النصى عند الإنتهاء من إستخدام جزء الذاكرة الذى يشير إليه pstring ، يتم تنفيذ الأمر التالى .

```
free(pstring);
```

دالة realloc

تمكنك دالة realloc من إعادة إستخدام جزء من الذاكرة قمت بحجزه مسبقاً ، و هذه الدالة تستقبل معاملين ، المعامل الأول مؤشر إلى مكان الذاكرة التى تريد إعادة إستخدامها ، و المعامل الثانى هو المساحة التى تريد حجزها ، و لا يمكنك حجز جزء أكبر من الجزء الذى تم حجزه فى المرة الأولى ، يمكنك فقط حجز مساحة مساوية أو أقل من المساحة التى تم حجزها فى المرة الأولى ، و إذا فعلت ذلك لن يتم حجز إلا المساحة التى حجزتها فى المرة الأولى.

مثال

يقوم هذا البرنامج بطباعة مجموعة من الأعداد ، لا يقوم المستخدم بتحديد عددها مُسبقاً.

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int sum = 0;
    int *pnum ;

    pnum = (int *)malloc(sizeof(int));

    printf("Enter the numbers , (0) to exit : ");

    while(*pnum != 0){
        realloc(pnum, sizeof(int));
        scanf("%d", pnum);
        sum += *pnum;
    }

    printf("Sum of numbers = %d", sum);

    return 0;
}
```

هنا يتم حجز جزء من الذاكرة مساحته مساحة متغير من النوع int (غالباً 4 بايت) ، ثم يقوم البرنامج بإستقبال الأرقام المدخل من المستخدم ، طالما ان القيمة المدخلة ليست صفراً ، حيث يقوم بإعادة إستخدام نفس الجزء من الذاكرة فى كل مرة لتخزين الرقم المدخل و ذلك عن طريق الدالة realloc ، و بلك نستطيع أن نجمع أى عدد من الأرقام فقط بإستخدام جزء من الذاكرة مساحته 4 بايت لتخزين كل هذه الأرقام !

سنكتفى فى هذا الفصل بالأمثلة المذكورة لشرح هذا الجزء من إستخدام المؤشرات ، و سنتناول استخدامات أخرى للمؤشرات فى أكثر من موضوع فى الفصول القادمة.

التمارين

1) اكتب برنامجاً يقوم بإستقبال مجموعة من الأرقام ، و إيجاد مجموعها ومتوسطها ، بإستخدام عملية حجز الذاكرة ديناميكياً ، بحيث لا يقوم المستخدم بتحديد عدد هذه الأرقام الذى سيقوم بإدخالها .

2) اكتب برنامجاً يقوم بإستقبال مجموعة غير محددة من الكلمات ، و يقوم بطباعتها مرة أخرى مرتبة حسب طولها بداية من الأقل طولاً إلى الأكثر طولاً .

3) اكتب برنامجاً يقوم بإستقبال نص معين ، ثم حذف أى مسافات أو علامات خاصة أو أرقام منه ، ثم طباعته . استخدم المؤشرات فى كل العملي التى ستقوم بها .

الفصل الثامن

الـــــــــــدوال

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ لماذا نستخدم الدوال ؟
- ✓ أنواع الدوال و كيفية تعريفها.
- ✓ الدوال التى ترجع بأكثر من خرج .
- ✓ الدوال التكرارية – Recursion .
- ✓ ما الفرق بين الدوال التكرارية و الحلقات التكرارية .

لماذا نستخدم الدوال؟

يجب أن تعلم أولاً أن هناك نوعين مختلفين من طرق البرمجة و تقسيم البرنامج، النوع الأول و هو المعتمد فى لغة السى و هو الـ structural programming و النوع الثانى هو الـ Object oriented programming ، و سأشرح لكم فى إيجاز ما هى البرمجية الهيكلية و المبادىء التى بنيت عليها. البرمجة الهيكلية تعتمد على تقسيم المشكلة إلى أجزاء صغيرة ، ثم حل كل مشكلة على حدة ، و فى النهاية تجميع هذه الحلول للحصول على الحل النهائى للمشكلة الأساسية . لذلك نقوم فى البرمجة الهيكلية بتقسيم برنامجنا إلى دوال كل دالة تقوم بعمل معين مستقل بذاته ، و هذا فى الحقيقة له مزايا عديدة للمبرمج. أولاً سهولة تطوير البرنامج و تنظيمه ، ثانياً إذا أردنا القيام بعملية معينة أكثر من مرة فى برنامجنا فلا حاجة لنا لكتابة الكود الخاص بها أكثر من مرة ، إنما فقط نقوم بإستدعاء الدالة التى تقوم بهذه المهمة ، ثالثاً من الممكن الإستفادة ببعض هذه الدوال الذى قمت بإنشائها فى برامج أخرى. و على الرغم أننا لم نتعامل سوى مع الدالة الرئيسية إلى الآن ، إلا أنك قد استخدمت البرمجة الهيكلية من حيث لا تدرى. تذكر كم مرة استخدمت فيها دوال جاهزة من الـ standard library ، سواء للطباعة و استقبال البيانات أو لإجراء العمليات على الـ strings و غيرهم. و من هنا إلى نهاية الكتاب بإذن اللّه سنقوم بتقسيم برامجنا إلى دوال ، و يجب عليك أنت أيضاً ذلك إذا أردت أن تبدأ أن تعمل

كمحترف و ليس كهاوى . نبدأ مع كيفية تعريف الدوال.

أنواع الدوال و كيفية تعريفها

الدالة فى أبسط صورها لا تأخذ أى معاملات و لا تعود بأى نتيجة و هو أول نوع من الدوال سنتناول شرحه، و نقوم بتعريف الدالة فى هذه الحالة كالآتى.

```
void اسم الدالة (){
    الجمل المراد تنفيذها
}
```

و void المكتوبة قبل إسم الدالة هى نوع الرجوع ، و هنا الدالة لا تعود بأى قيمة لذا استخدمنا void.

و فى النوع الثانى من الدوال الذى سنتناول شرحه تعود الدالة بقيمة و يتم وضع نوع هذه القيمة قبل اسم الدالة ، مثال الدالة الرئيسية.فهى تعود بـ **0** ، و هى قيمة من النوع int ، لذلك تم وضع int قبل اسم الدالة.

```
#include <stdio.h>

int main(){

    الجمل المراد تنفيذها

    return 0;

}
```

أما النوع الثالث فتستقبل فيه الدالة معاملات و تعود بقيمة ،مثال توضيحى.

مثال

يقوم هذا البرنامج بجمع رقمين يقوم المستخدم بإدخالهم، و لكن بإستخدام دالة خاصة تقوم بعملية الجمع.

```
#include <stdio.h>

int add_two_nums(int x, int y){

    int res = x + y;

    return res;
}

int main(){

    int num1, num2, sum = 0;

    printf("Enter two numbers: ");
    scanf("%d%d", &num1, &num2);

    sum = add_two_nums(num1, num2);
    printf("sum = %d", sum);

    return 0;
}
```

شرح البرنامج

الدالة الرئيسية تستقبل البيانات من المستخدم ، ثم يتم إستدعاء دالة الجمع و إمرار قيمتى المتغيرين num1 و num2 إليها ، فيقوم البرنامج بالذهاب إلى دالة الجمع أولاً لتنفيذها قبل أى يكمل تنفيذ الدالة الرئيسية ، فيقوم فى البداية بحفظ القيمتين فى متغيرين x و y ، ثم يقوم بجمعهم و حفظ النتيجة فى res و يعود بقيمتها . هنا يعود البرنامج إلى استكمال تنفيذ الدالة الرئيسية بعد الإنتهاء من تنفيذ الدالة . فيقوم بحفظ القيمة التى عادت بها الدالة فى متغير sum ثم يقوم بطباعة النتيجة .

إختبار البرنامج

```
Enter two numbers: 100 200
sum = 300
Process returned 0 (0x0)   execution time : 3.331 s
Press any key to continue.
_
```

البرنامج يعمل بالشكل المُتوقع .

لعلك لاحظت أنه تمت كتابة دالة الجمع قبل الدالة الرئيسية ، و لكننا لا نفعل ذلك عادة ، إنما نقوم بكتابة أى دالة بعد الدالة الرئيسية ، و لكن سيظهر لنا مشكلة . هل يمكن تنفيذ دالة فى الدالة الرئيسية قبل تعريفها أصلاً ؟ .. الدالة مثلها فى ذلك مثل أى متغير . هل يمكن أن نستخدم متغير قبل تعريفه أصلا ، فمثلاً نقوم بتعديل المثال السابق كالتالى ؟ .

```
14      printf("Enter two numbers: ");
15      scanf("%d%d", &num1, &num2);
16
17      int num1, num2, sum = 0;
```

بالطبع لا ، و هذا هو نوع الخطأ الذى سيظهر لنا.

```
C:\Users\mahmo...  15    error: 'num1' undeclared (first use in this function)
C:\Users\mahmo...  15    note: each undeclared identifier is reported only once for each function it appears in
C:\Users\mahmo...  15    error: 'num2' undeclared (first use in this function)
```

الخطأ هو أننا استخدمنا num1 و num2 قبل تعريفهم أصلاً . و نفس الشىء مع الدوال لا يمكن إستخدامها قبل تعريفها ، لذلك نقوم بكتابة ما يعرف بالـ prototypes قبل الدالة الرئيسية ، و هى بمثابة إخطار للبرنامج إن هذه الدالة تم تعريفها بعد الدالة الرئيسية أو بعد الدالة التى تم إستدعائها منها سواء تم إستدعاؤها من الدالة الرئيسية أو دالة أخرى. فبإستخدام الـ prototypes يكون شكل المثال السابق كالآتى.

```
#include <stdio.h>

int add_two_nums(int x, int y);

int main(){
    int num1, num2, sum = 0;
    printf("Enter two numbers: ");
    scanf("%d%d", &num1, &num2);
    sum = add_two_nums(num1, num2);
    printf("sum = %d", sum);
    return 0;
}

int add_two_nums(int x, int y){
    int res = x + y;
    return res;
}
```

هنا دالة الجمع معرفة بداية من الـ prototype الخاص بها حتى دالة الجمع ، أى دالة أخرى و ضعت فى هذا المدى يمكنها إستدعاء دالة الجمع . السؤال هنا ماذا لو أن هناك دالة أخرى بعد دالة الجمع هل يمكنها إستدعاء دالة الجمع ؟ .. الإجابة لا ، لأنها لا تقع فى المدى الذى تم تعرفها فيه { بداية الـ prototype حتى الدالة } .

يمكنك عدم كتابة اسم متغيرات المعاملات فى الـ prototype ، أو كتباتها بإسم يختلف عن إسمها فى رأس تعريف الدالة . فيمكن كتابتها كذلك .

```
int add_two_nums(int , int );
```

أو هكذا.

```
int add_two_nums(int z, int n);
```

و لكنه لا بد لك من كتابة نوع هذه المعاملات، فلا يمكنك هنا عدم كتابة int .

و هنا نكون قد انتهينا من الجزء الأول من هذا الفصل، و نكون قد تناولنا حتى الآ 3 أنواع من الدوال:

1) دالة لا تعود بقيمة و لا تأخذ معاملات.
2) دالة تعود بقيمة واحدة و لا تأخذ معاملات.
3) دالة تعود بقيمة واحدة و تأخذ معاملات.

و سنتناول فى الجزء الثانى من هذا الفصل النوع الرابع من الدوال و هى الدوال التى تعود بأكثر من قيمة ،و هو نوع هام جداً من الدوال و يعتبر من المواضيع المتقدمة فى اللغة . سنتناوله معاً بشكل مبسط بإذن اللّه.

ماذا نفعل إذا أردنا أن نعود بأكثر من خرج من الدالة ، و أقصى خرج للدالة عن طريق return هو قيمة واحدة ؟ ببساطة يتم ذلك عن طريق إستخدام المؤشرات كمعاملات للدالة. المثال التالى يوضح كيفية فعل ذلك.

مثال

يقوم هذا البرنامج بإستقبال رقم من المستخدم، ثم يقوم بطباعة إشارة هذا الرقم و القيمة الصحيحة له و قيمة الكسر الموجود فيه.

```
#include <stdio.h>
#include <math.h>

void seperate(double num, char *psign, int *pwhole, double *pfraction);

int main(){

    double num,fraction;
    char sign;
    int whole;

    printf("Enter the number: ");
    scanf("%lf", &num);

    seperate(num, &sign, &whole, &fraction);

    printf("sign: %c\nwhole num: %d\nfraction: %.2f", sign, whole, fraction);

    return 0;
}
```

```c
void seperate(double num, char *psign, int *pwhole, double *pfraction){

    double magnitude;

    if(num > 0)
        *psign = '+';
    else if(num < 0)
        *psign = '-';
    else
        *psign = ' ';

    magnitude = fabs(num);
    *pwhole = floor(magnitude);
    *pfraction = magnitude - *pwhole;
}
```

شرح البرنامج

الدالة الرئيسية قمنا فيها بإستقبال الرقم من المستخدم ثم تخزينه فى num ، و كذلك قمنا بتعريف متغيرات لحفظ أجزاء الرقم بعد عملية الفصل، ثم قمنا بإستخدام دالة separate للفص، ثم قمنا بطباعة كل جزء.

أما بالنسبة لدالة separate , فهى تستقبل 4 معاملات ، num الرقم المُدخل، و psign مؤشر يشير إلى المتغير sign الذى تم تعريفه فى الدالة الرئيسية ، و بالمثل فأن pwhole يشير إلى المتغير whole و pfraction يشير إلى المتغير fraction .

المعامل الأول num يعمل كدخل للدالة ، أما باقى المعاملات فيعملون كخرج و ليس كدخل لأننا نستخدم هذه المؤشرات فى تغيير قيم المتغيرات الموجودة فى الدالة الرئيسية. إذاً فهذه الدالة ذات 1 دخل و هو الرقم ، و 3 خرج و هم نتائج عملية الفصل.

عملية الفصل تتم كالآتى ، يتم إختبار الرقم المدخل و على حسب قيمته تكون الإشارة و هذا يتم عن طريق جملة if الشرطية . ثم نقوم بحساب القيمة المطلقة للرقم و تخزينها فى المتغير magnitude ، و يتم هذا عن طريق الدالة fabs الموجودة فى math.h ، ثم نقوم بإيجاد القيمة الصحيحة للرقم عن طريق الدالة floor و حفظها فى whole عن طريق المؤشر pwhole ثم نقوم بإيجاد قيمة الكسر عن طريق عملية طرح بين القيمة المطلقة للرقم و قيمة المتغير whole (أو قيمة المتغير الذى يشير عليه المؤشر pwhole) ، و هنا تنتهى دالة الفصل seperate.

إختبار البرنامج

```
Enter the number: -5.3
sign: -
whole num: 5
fraction: 0.30
Process returned 0 (0x0)   execution time : 3.594 s
Press any key to continue.
```

البرنامج يعمل بالشكل المُتوقع .

الآن و قد تعلمنا كيفية إستخدام معامل الدالة كخرج لها ، سنقوم بعمل برنامج نستخدم فيه معامل واحد لدالة معينة يعمل كدخل و خرج فى نفس الوقت .

مثال

يقوم البرنامج بمهام الآلة الحاسبة و لكن بطريقة تختلف لما تعرضنا إليه سابقاً ، سيقوم البرنامج بإستقبال رمز العملية المراد تنفيذها و الرقم الثانى من المستخدم ،و يتم إجراء العملية على ناتج العملية السابقة مع الرقم الثانى . عملية استقبال البيانات تتم عن طريق دالة scan_data ، و تنفيذ العملية المختارة يتم عن طريق دالة do_next_op . و هذا خرج يوضح كيفية عمل البرنامج .

```
#include <stdio.h>

void scan_data(char *pop, double *poperand);
void do_next_op(char *pop, double *poperand, double *paccum);

int main(void){

    char op;
    double operand, accum = 0;

    do{
    scan_data(&op, &operand);
    do_next_op(&op, &operand, &accum);
    printf("result so far = %.2f\n", accum);
    }while(op != 'q');

    printf("final result = %.2f", accum);
    return 0;
}
```

```
void scan_data(char *pop, double *poperand){
    scanf(" %c%lf", pop, poperand);
}

void do_next_op(char *pop, double *poperand, double *paccum){

    switch(*pop){
    case '+':
        *paccum += *poperand;
        break;
    case '-':
        *paccum -= *poperand;
        break;
    case '*':
        *paccum *= *poperand;
        break;
    case '/':
        *paccum /= *poperand;
        break;
    case 'q':
        break;
    default:
        printf("this operation is not valid");
    }
}
```

شرح البرنامج

```
#include <stdio.h>

void scan_data(char *pop, double *poperand);
void do_next_op(char *pop, double *poperand, double *paccum);
```

الدالة scan_data تستقبل مؤشرين أحدهما يشير إلى نوع العملية و الآخر يشير إلى المعامل الثاني للعملية، و الدالة do_next_op تستقبل نفس المؤشرين إضافة إلى مؤشر يشير إلى ناتج العملية السابقة .

```
int main(void){

    char op;
    double operand, accum = 0;

    do{
    scan_data(&op, &operand);
    do_next_op(&op, &operand, &accum);
    printf("result so far = %.2f\n", accum);
    }while(op != 'q');

    printf("final result = %.2f", accum);
    return 0;
}
```

يتم إعطاء قيمة إبتدائية لناتج العملية السابقة بصفر . الحلقة التكرارية do-loop تقوم بتكرار البرنامج طالما أن المتغير op - العملية التى يريد أن يقوم بها المستخدم – لا تساوى q. الحلقة تقوم بإستدعاء دالة scan_data و إمرار عنوان متغيرين نوع العملية و المعامل ، ليتم إستقبال البيانات من المستخدم و حفظهما فيهم. ثم نقوم بإستدعاء الدالة do_next_op لتقوم بالقيام بالعملية الحسابية، و نلاحظ أن المؤشر paccum يستخدم كدخل للدالة قبل العملية الحسابية و كذلك يستخدم كخرج منها بعد العملية الحسابية ليتم إستخدامه عند إستدعاء الدالة مرة أخرى كدخل لها – كناتج العملية السابقة، ثم يتم طباعة الناتج الحالى . فى حالة إدخال q و أى رقم ، سيتم الخروج من الحلقة و طباعة الناتج النهائى .

```
void scan_data(char *pop, double *poperand){
    scanf(" %c%lf", pop, poperand);
}
```

الدلة scan_data فقط تستقبل البيانات من المستخدم عن طريق scanf .

```
void do_next_op(char *pop, double *poperand, double *paccum){

    switch(*pop){
    case '+':
        *paccum += *poperand;
        break;
    case '-':
        *paccum -= *poperand;
        break;
    case '*':
        *paccum *= *poperand;
        break;
    case '/':
        *paccum /= *poperand;
```

```
            break;
        case 'q':
            break;
        default:
            printf("this operation is not valid");
        }
}
```

الدالة do_next_op تقوم بإجراء العملية المناسبة بإستخدام switch case ، فمثلا فى عملية الجمع يتم جمع ناتج العملية السابقة و المعامل المدخل من المستخدم ، وحفظهما فى ناتج العملية السابقة حيث سيتم إستخدامها عند تكرار إستدعاء الدالة كدخل لها.

الدوال التكرارية – Recursion functions

ما هى الدالة التكرارية ؟

الدالة التكرارية هى دالة تقوم بإستدعاء نفسها، أو تقوم بإستدعاء دالة أخرى تقوم بطريقة مباشرة أو غير مباشرة بإستدعاء نفس الدالة الأولى.

مثال

سنقوم بعمل برنامج يقوم بحساب مضروب الرقم ، و لكن بدلاً من إستخدام حلقة تكرارية ، سنقوم بإستخدام الدوال التكرارية – Recursion functions .

```
#include <stdio.h>

double fact(int num);

int main(){

    int num;
    double f;

    printf("Enter the number: ");
    scanf("%d", &num);

    f = fact(num);

    printf("factorial = %.2f", f);

    return 0;
}
```

```
double fact(int num){

    double res = 0;

    if(num == 1 || num == 0)
        res = 1;
    else
        res = num * fact(num - 1);

    return res;
}
```

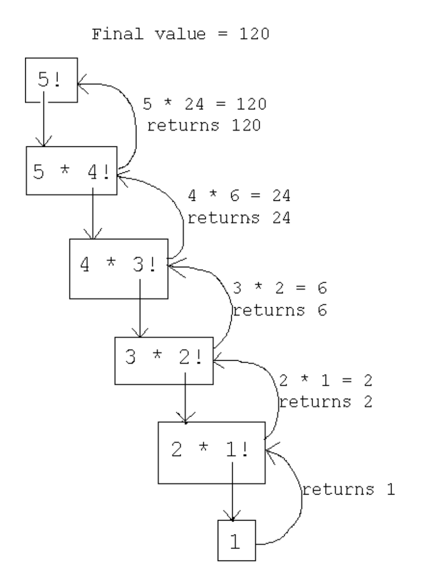
شرح المثال

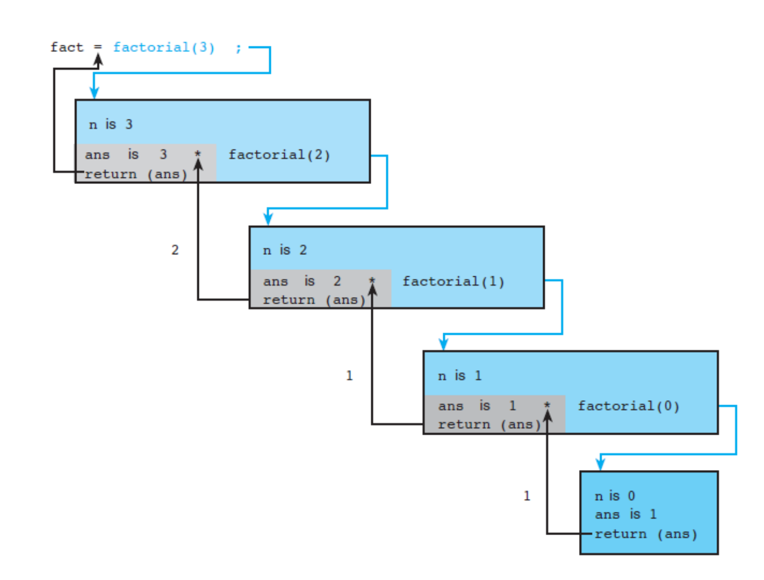
البرنامج يقوم بإستقبال رقم من المستخدم ثم حساب مضروب هذا الرقم عن طرق الدالة fact التى تستقبل هذا الرقم كدخل لها ، و تعود بمضروبه الذى يتم حفظه فى المتغير f .

فى أى دالة تكرارية يتم إستخدام جملة شرطية لإنهاء تكرار الدالة عند تحقق هذا الشرط ، الدالة هنا تتوقف عن تكرار نفسها عندما يكون num=1 أو num=0 و تعود بقيمة المضروب الخاص بهما و هو 1 . أما إذا كان الـ num أكبر من الواحد ، فيتم حساب المضروب عن طريق ضرب الرقم فى مضروب (الرقم -1) ، فمثلاً مضروب رقم 5 يساوى(5 * مضروب 4) ، و يتم حساب مضروب الـ 4 عن طريق إستدعاء الدالة fact و إعطائها الرقم 4 ، و عند حساب مضروب الـ 4 سيتم حسابه عن طريق ضرب رقم 4 فى مضروب الرقم 3

... و هكذا حتى تكون الـ num = 1 فتعود الدالة بـ 1 .

صور توضيحية لكيفية عمل الدالة التكرارية السابقة .





اختبار البرنامج

```
Enter the number: 5
factorial = 120.00
Process returned 0 (0x0)   execution time : 2.451 s
Press any key to continue.
```

البرنامج يعمل بالشكل المُتوقع .

مثال

سنتناول مثال الآخر، و لكن هذه المرة سنتعامل مع الـ strings لشهرة إستخدامها مع الدوال التكرارية

يقوم هذا البرنامج بإيجاد عدد تكرار حرف فى كلمة معينة بإستخدام الدوال التكرارية .

```c
#include <stdio.h>

int count(char ch,char *str);

int main(){

    char s[] = "mahmoud";

    int c = count ('m', s);

    printf("number of m = %d", c);

    return 0;
}

int count(char ch, char *str){

    int ans;

    if(str[0] == '\0')
        ans = 0;
    else
        if(ch == str[0])
            ans = 1 + count(ch, &str[1]);
        else
            ans = count(ch, &str[1]);

    return ans;
}
```

الدالة count تستقبل الحرف المراد إيجاد عدده ، و مؤشر إلى عنوان بداية الكلمة المراد البحث. شرط توقف تكرار الدالة هنا أن يكون الحرف الأول 0\ ، و حينها يكون العدد =0 ، و فى حالة عدم تحقق هذا الشرط تختبر الجملة الشرطية عما إذا كان الحرف الأول من الكلمة مساوياً للحرف المراد حساب عدد مرات تكراره أم لا ، فى حالة إذا كان مساوياً يتم إضافة 1 على العداد ثم تكرار استدعاء الدالة و لكن بكلمة جديدة بدايتها str[1] أى أن الكلمة التى ستوضع تحت الإختبار هى ahmoud و التكرار الثالث للدالة ستكون hamoud، و هكذا حتى تنتهى الكلمة فيكون أول حرف 0\ فيتوقف تكرار الدالة.

اختبار البرنامج

```
number of m = 2
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

و بالفعل عدد تكرار حرف m فى كلمة mahmoud هو مرتان ، و هذا يؤكد عمل البرنامج بشكل صحيح . يمكنك تجربة البرنامج بإستخدام أى كلمة أو حرف آخرين.

الفرق بين الدوال التكرارية و الحلقات التكرارية

و لعلك لاحظت أن الدوال التكرارية تقوم بنفس عمل الحلقات التكرارية تقريباً، فأى واحدة سنستخدم فى برامجنا ؟ .. فى أغلب الأحيان سنستخدم الحلقات التكرارية لأنها توفر فى مساحة الـ stack (جزء من الـ RAM) و كذلك أسرع فى التنفيذ من الدوال التكرارية، و لكن فى بعض الأحيان القليلة تقدم الدوال التكرارية المشكلة فى صورة أكثر بساطة من الحلقات التكرارية. لذلك لم أرد أن أفرد فصلاً خاصاً بالدوال التكرارية و اكتفيت بعرض الفكرة الأساسية لها .

سيتم الإكتفاء فى هذا الفصل بالأمثلة التى تم شرحها و بإذن الله سيتم التطبيق على الدوال فى البرامج التطبيقية للفصول القادمة. أترككم الآن مع التمارين.

التمارين

1) اكتب برنامجاً يقوم بتحديد الرقم الأكبر و الأصغر من بين 3 ارقام يقوم بإدخالهم المستخدم ، قم بإستخدام دالتين ،الأولى تقوم بإستقبال 3 مؤشرات للأرقام الثلاثة – الأرقام الثلاثة يتم إستقبالهم و حفظهم فى متغيرات بالدالة الرئيسية - ، ثم تعود بالقيمة الأقل ، و الثانية تقوم بإستقبال 3 مؤشرات للأرقام الثلاثة، ثم تعود بالقيمة الأكبر.

2) الأرقام المثلثية هى أرقام تنتج من جمع الأرقام الصحيحة، فمثلاً الرقم المثلثى السابع = 1 + 2 + 3 + 4 + 5 + 6 +7 أى أنه = 28 . فتكون أول 10 أرقام مثلثية هى :

1 , 3 , 6 , 10 , 15 , 21 , 28 , 36 , 45 , 55 , ...

معاملات أول 7 أرقام مثلثية كالآتى :

1: 1

3: 1,3

6: 1,2,3,6

10: 1,2,5,10

15: 1,3,5,15

21: 1,3,7,21

28: 1,2,4,7,14,28

نلاحظ أن رقم 28 هو أول رقم مثلثى تتجاوز عدد معاملاته 5 معاملات ، إذن أن عدد معاملاته 6 معاملات .

فاكتب برنامجاً يقوم بإيجاد أول رقم مثلثى تتجاوز عدد معاملاته 10 معاملات ، مستعيناً بدالتين.الدالة الأولى تستقبل رتبة الرقم المثلثى المراد إيجاده ، و تعود بقيمته . و الدالة الثانية تستقبل هذا الرقم و تقوم بإيجاد عدد معاملاته .

3) اكتب برنامجاً يستخدم دالتين الأولى تقوم بتحويل الرقم العشرى (Decimal) إلى ثنائى (Binary) ، و الثانية تقوم بتحويل الرقم الثنائى إلى رقم عشرى ، المستخدم هو من يقوم بإدخال الرقم المراد تحويله فى كلتا الحالتين .

4) اكتب برنامجاً يقوم بحساب المعامل المشترك الأكبر بإستخدام الدوال التكرارية (Recursion) ,

الفصل التاسع

التعامل مع الملفات

ما يجب أن تكون قد تعلمته فى نهاية هذا الفصل ؟

- ✓ لماذا نستخدم الملفات ؟
- ✓ ما هى الأنواع المختلفة للملفات ،و ما الفرق بينها ؟
- ✓ الدوال التى تستخدم فى العمليات على الملفات.
- ✓ التعامل مع الملفات النصية – text files .
- ✓ التعامل مع الملفات الثنائية – binary files .

لماذا نستخدم الملفات ؟

فى برامجنا السابقة كنا نستقبل البيانات من المستخدم و يتم حفظها مؤقتاً فى الذاكرة المؤقتة RAM و عند إغلاق البرنامج نفقد هذه البيانات فعندما نفتح البرنامج مرة أخرى لإستخدام هذه البيانات لا نستطيع لأنها فُقدت، لذلك يجب إيجاد طريقة يتم فيها الإحتفاظ بالبيانات بصفة دائمة لذلك سنستخدم الهارد ديسك لحفظ هذه البيانات لإستخدامها فى أى وقت حتى بعد إغلاق البرنامج و سيتم حفظ هذه البيانات على الهارديسك فى ملف – file .

العمليات الأساسية على الملفات

نبدأ مع مجموعة من العمليات الأساسية التى تستخدم عند التعامل مع الملف أياً كان نوعه –و سنتعرض لاحقاً فى هذا الفصل إلى أنواع المفات التى سنتعامل معها.

فتح الملف

يتم فتح أى ملف بإستخدام هذه الدالة

```
fopen(اسم الملف , "العملية التى ستقوم بها");
```

اسم الملف هنا عبارة عن مؤشر يشير إلى مكان الملف على الهارد ديسك .

و العملية التى ستقوم بها على الملف توضع بين علامتى تنصيص، و هذا الجدول يحتوى على أهم أنواع العمليات على الملفات .

العملية	الرمز المستخدم
الكتابة	"w"
القراءة	"r"
الإضافة	"a"

تلاحظ من الجدول السابق أننا إذا أردنا ان نفتح ملف للكتابة نستخدم "w" ، و إذا أردنا أن نفتحه لقراءة بيانات نستخدم "r" ، و إذا أردنا أن نضيف بيانات على البيانات الموجدة فى الملف نستخدم "a" و الفرق بين إضافة بيانات إلى ملف و كتاب بيانات إلى ملف ، أن الإضافة تكون بعد البيانات القديمة الموجودة فى الملف – إن

وجدت – و فى حالة الكتابة يتم حذف القديم و كتابة بيانات جديدة ، و هذه العمليات يتم إستخدامها مع الملفات النصية – text files ، و ليس مع الملفات الثنائية – binary files ، و سنتعرض للنوعين فى هذا الكتاب و سيتم شرح الفرق بينهما لاحقاً فى هذا الفصل بشىء من التفصيل. أما إذا أردت أن تقوم بهذا العمليات مع الملفات الثنائية . فهذا الجدول يوضح لك الرموز المستخدمة للعمليات على هذا النوع من الملفات.

العملية	الرمز المستخدم
الكتابة	"wb"
القراءة	"rb"
الإضافة	"ab"

ستلاحظ أنه يتم إضافة b فقط رمزاً إلى binary .

فى الأمر التالى تم فتح ملف للكتابة .

```
fopen("D:\\C\\C files\\mytext.txt" , "w");
```

يجب عليك أن تضع عنوان الملف الذى تريد كتابة البيانات إليه بالشكل السابق ، و لاحظ أنه تم استخدام \\ و ليس \ واحدة ، لأنه إذا وضعت \ سيعتقد المترجم أنك تريد أن تكتب escape sequence مثل \n .

دالة fopen تعود بمكان الملف فى الذاكرة ، و سنستخدم هذا المؤشر فى عمليات أخرى فى أثناء تعاملنا مع الملف لذا يتم حفظ هذا العنوان فى مؤشر من النوع FILE كالآتى .

```
FILE *pfile = NULL;

pfile = fopen("D:\\C\\C files\\mytext.txt" , "w");
```

يتم إعطاء المؤشر قيمة إبتدائية NULL كإجراء وقائى لا أكثر . دالة fopen إذا لم تجد الملف فى هذا المكان ستقوم بإنشاؤه لك، فمنها يمكنك إنشاء ملف جديد كذلك.

إغلاق الملف

بعد الإنتهاء من العمل مع الملف يتم غلقه عن طريق الدالة التالية.

```
fclose(pfile);
```

يتم إعطاء الدالة مؤشر يشير إلى مكان الملف فى الذاكرة و هو pfile الذى استخدمناه فى هذا المثال ، و لعلك تعلم الآن الهدف من وراء حفظ هذا المؤشر عند فتح الملف.

إذا أردت أن تكتب "ثم" تقرأ من ملف ، فيجب عليك فتحه للكتابة ثم " غلقه" ثم فتحه مرة أخرى للقراءة .

حذف الملف

إذا أردت أن تحذف الملف بعد الإنتهاء منه يمكنك إستخدام الدالة التالية.

```
remove("D:\\C\\C files\\mytext.txt");
```

دالة الحذف تأخذ – مقل دالة فتح الملف- مؤشر يشر إلى عنوان الملف فى الهارد ديسك .

و هنا نكون قد انتهينا من أهم العمليات التى تستخدم مع الملفات، سنتطرق الآن إلى أنواع الملفات بشىء من التفصيل و كيفية التعامل مع كلاً منها على حدة.

أنواع الملفات

النوع الأول من الملفات هو الملفات النصية التى استخدمناها فى توضيح الأمثلة السابقة، و فيها يتم تحويل البيانات المدخلة إلى الذاكرة أى يتم تحويلها إلى الهيئة الثنائية Binary لأنه كما نعلم أن الذاكرة تحفظ البيانات على الشكل الثنائى ، ثم يتم تحويله إلى نصى مرة أخرى لوضعه فى الذاكرة.

أما النوع الثانى و هو الثنائى يتم وضع البيانات المدخلة فى الذاكرة ثم وضعها كما هى فى الملف على صورتها الثنائية ، أى أنه لا تتم هنا عملية تحويل مرة أخر للبيانات.

لذا فنحن نفضل فى معظم الأحيان إستخدام الملفات الثنائية لسرعتها و سهولة التعامل معها، و هى النوع التى سنركز أكثر عليه فى تعاملاتنا مع الملفات ، و إن كنا سنتعامل مع النوعين و سنتناول طرق إجراء العمليات عليهم .

الملفات النصية

تستخدم مع الملفات النصية أكثر من نوع من أنواع الدوال التى تستخدم للقراءة و الكتابة من الملفات، منها من يختص بالحروف فقط ، و منها من يختص بالـ srtings فقط ، و منها من يتعامل مع جميع أنواع البيانات و هذا النوع هو من سنتطرق لشرحه هنا .

كتابة بيانات إلى ملف

لكتابة مجموعة من البيانات إلى ملف نستخدم دالة تطرقنا إلى إستخدام توأمها من قبل إلا أن الفرق أنه هذه تتعامل مع الملفات و ليس المستخدم ألا و هى دالة fprintf(). هذه الدالة لها نفس استخدام و كيفية عمل printf() تقريباً إلا أنها تستقبل معامل جديد يوضع فيه مؤشر يشير إلى مكان الملف فى الذاكرة.

فى المثال التالى يتم إدخال مجموعة من البيانات إلى ملف نصى عن طريق fprintf().

```
int age = 21;

fprintf(pfile, "my age = %d", age);
```

لاحظت استخدام fprintf() تماماً كإستخدام printf() ، و لكن يتم وضع مؤشر للملف كأول معامل .

قراءة بيانات من ملف

لقراءة مجموعة من البيانات من ملف نستخدم دالة ؟ .. نعم ، fscanf() و على نفس الشاكلة و نفس الفروق مع scanf().

مثال

يقوم البرنامج بإستقبال 3 درجات ، ثم حفظها فى ملف نصى، ثم قرائتها مرة أخرى من الملف و حفظها فى متغيرات أخرى ، وأخيراً حساب المتوسط و طباعته.

```c
#include <stdlib.h>
#include <stdio.h>

int main(){

    FILE *pfile = NULL;
    char *filename = "D:\\C\\C files\\mytext.txt";
    int grade1, grade2, grade3,
        fgd1, fgd2, fgd3, sum = 0;
    float avg;

    printf("Enter 3 grades: ");
    scanf("%d%d%d", &grade1, &grade2, &grade3);

    if(!(pfile = fopen(filename, "w"))){
        perror("can not open file for writing");
        exit(1);
    }

    fprintf(pfile, "%d %d %d", grade1, grade2, grade3);
    fclose(pfile);

    if(!(pfile = fopen(filename, "r"))){
        perror("can not open file for reading");
        exit(1);
    }

    fscanf(pfile, "%d%d%d", &fgd1, &fgd2, &fgd3);
    fclose(pfile);

    sum = fgd1 + fgd2 + fgd3;
    avg = sum / 3;

    printf("average = %.2f", avg);

    return 0;
}
```

شرح المثال

```c
FILE *pfile = NULL;
char *filename = "D:\\C\\C files\\mytext.txt";
```

هنا تم تعريف مؤشر من النوع FILE ليشير إلى مكان الملف فى الذاكرة ، و كذلك مؤشر من النوع char ليشير إل مكان الملف فى الهارد ديسك، و هنا تم حفظ المكان فى متغير لأننا سنستخدمه فى أكثر من عمل فى البرنامج ، لذا يفضل فعل ذلك ى كل برنامج لك لتوفير جهد كتابة العنوان كاملاً عندما تحتاجه فى كل عملية.

ثم تم تعريف 3 متغيرات لإستقبال البيانات من المستخدم فيهم ، و 3 آخرين عند القراءة من الملف، و 2 آخرين لمجموع الدرجات و المتوسط، و تم إستقبال البيانات من المستخدم بعدها.

```
if(!(pfile = fopen(filename, "w"))){
    perror("can not open file for writing");
    exit(1);
}
```

هنا جملة شرطية يتم فى شرطها فتح الملف للكتابة بالصورة التى تعودنا عليها، و هذه الجملة الشرطية تختبر عما إذا تم فتح الملف بالفعل أم أن هناك خطأ ما حدث لم يمكنه من فتح الملف، ففى حالة عدم فتح الملف لأى خطأ حدث، تعود دالة fopen() بـ NULL فيتم طباعة الجملة الموضحة للمستخدم مع سبب هذا الخطأ و الذى تتكفل بإيجاده دالة perror() ، ثم يتم إستخدام دالة exit() و تستخدم لإخطار نظام التشغيل بأن البرنامج لم يسير بالشكل الطبيعى المتوقع و هى موجودة فى stdlib.h لذا تم إستيرادها فى بداية البرنامج ، و يجب أن تستخدم هذه الجملة الشرطية عند فتح أو حذف ملف للتأكد من عدم حدوث أخطاء غير متوقعة.

```
fprintf(pfile, "%d %d %d", grade1, grade2, grade3);
fclose(pfile);
```

و بعدما تم فتح الملف للكتابة، نقوم بإدخال البيانات بالصورة التى استخدمناها من قبل فى الأمثلة ، ثم غلق الملف.

```
if(!(pfile = fopen(filename, "r"))){
    perror("can not open file for reading");
    exit(1);
}

fscanf(pfile, "%d%d%d", &fgd1, &fgd2, &fgd3);
fclose(pfile);
```

و بنفس الطريقة تم فتح الملف للكتابة ثم استقبال البيانات منه ثم إغلاقه. ثم يتم بعد ذلك حساب المتوسط و طباعته للمستخدم، و إذا ذهبت لفتح الملف بعد إنتهاء البرنامج من المكان الذى حفظته فيه ، ستجد أن الدرجات ما زالت محفوظة فى الملف، و لم تذهب بمجرد إغلاق البرنامج، و هذا هو سبب إستخدامنا للملفات فى المقام الأول .

نكون الآن قد انتهينا من دراستنا للملفات النصية، و سنتناول الآن الملفات الثنائية- binary files .

الملفات الثنائية (Binary Files)

تتميز الملفات الثنائية –إضافة إلى سرعتها- بسهولة عملية الكتابة و القراءة منها ، و سنتطرق إلى الدوال المستخدمة فى قراءة و كتابة البيانات منها و إليها.

كتابة البيانات إلى ملف

يتم كتابة البيانات إلى الملفات الثنائية عن طريق الدالة التالية .

```
FILE *pfile = NULL;
char *filename = "D:\\C\\C files\\myfile.bin";
int grades[] = {10, 20, 30};

if(!(pfile = fopen(filename, "wb"))){
    perror("can not open file for writing");
    exit(1);
}

fwrite(grades, sizeof(int),3 , pfile);
fclose(pfile);


return 0;
```

فى هذا المثال نقوم بفتح ملف ثنائى ثم كتابة مصفوفة مكونة من 3 درجات إليه ، و ذلك عن طريق fwrite().
المعامل الأول لهذه الدالة هو العنوان الذى تبدأ من عنده الدالة القراءة ، و نعلم أن اسم المصفوفة يعود بعنوان أول عنصر فيها. أما المعامل الثانى فهو حجم كل عنصر فيها، و الثالث هو عدد العناصر، و الرابع هو مؤشر الملف.

قراءة البيانات من ملف

يتم قراءة البيانات من الملفات الثنائية عن طريق دالة fread() ، و هى تعمل بنفس طريقة fwrite() ، إلا أن المعامل الأول هو عنوان ما سيقرأ إليه من الملف و ليس ما سيكتب منه إلى الملف.

تعديل المثال السابق ليقرأ ما كُتب فى الملف إلى مصفوفة أخرى ثم طباعتها .

```c
  FILE *pfile = NULL;
  char *filename = "D:\\C\\C files\\myfile.bin";
  int grades[] = {10, 20, 30};
  int fgd[3];

  if(!(pfile = fopen(filename, "wb"))){
      perror("can not open file for writing");
      exit(1);
  }

  fwrite(grades, sizeof(int),3 , pfile);
  fclose(pfile);

    if(!(pfile = fopen(filename, "rb"))){
        perror("can not open file for writing");
        exit(1);
    }

    fread(fgd, sizeof(int),3 , pfile);
    fclose(pfile);

    for(int i = 0; i < 3; i++)
        printf("%d\n", fgd[i]);

    return 0;
}
```

و هنا سيكون خرج البرنامج بنفس عناصر المصفوفة الأولى، و هذا يدل على أن عملية القراءة و الكتابة من و إلى الملف تمت كما نريد ، الخرج كالآتي.



هنا نكون قد انتهينا من شرح كلا من نوعى الملفات و العمليات المستخدمة معهما، و سنتناول الآن برنامج عملى لنوظف فيه ما تعلمناه ، و نتعلم بعض المهارات عند التعامل مع الملفات.

برنامج تطبيقى

سنقوم بتصميم برنامج بإستقبال بيانات الموظفين من المستخدم و حفظها فى ملف ، و به إمكانية البحث عن إسم موظف معين و عرض بياناته، أو إضافة بيانات موظف جديد، أو عرض بيانات جميع الموظفين.

```
1   /* employees' database program
2    * by: Mahmoud Elbarbary
3    * date : 11/4/2014
4    */
5
6   #include <stdio.h>
7   #include <stdlib.h>
8   #include <string.h>
9   #include <ctype.h>
10  #include <stdbool.h>
11  #define NAME_MAX 20
12
13  struct{
14      char *filename;
15      FILE *pfile;
16  }global = {"D:\\C\\C files\\mytext5.bin", NULL};
17
18  typedef struct employee{
19      char first_name[NAME_MAX];
20      char last_name[NAME_MAX];
21      int gross_income;
22      char martial_status[NAME_MAX];
23  }Employee;
24
25  bool get_person(Employee *pemployee);              /* Input function */
26  void getname(char *name);                          /* Read a name */
27  void show_person_data(void);                       /* Output function */
28  void record_search(void);
29
30  int main(void){
31
32      Employee member;
33
34      printf("\n%22c---------------------------\n", ' ');
35      printf("%22cWelcome to Database Program\n", ' ');
36      printf("%22c---------------------------\n\n", ' ');
37
38      if(!(global.pfile = fopen(global.filename, "wb"))){
39          perror("can not open file to write");
40          exit(1);
41      }
42
```

```
42
43      while(get_person(&member))
44          fwrite(&member, sizeof member, 1, global.pfile);
45
46      fclose(global.pfile);
47      show_person_data();
48
49      if(remove(global.filename)){
50          perror("can not delete the file");
51          exit(1);
52      }
53
54      return 0;
55  }
57  bool get_person(Employee *temp){
58
59      static char more = '\0';
60
61      printf("%26cSELECT AN OPERATION\n\n", ' ');
62      printf("%5c(A: adding an employer, R: retreiving data,"
63              "S: search for record)\n\n%35c", ' ',' ');
64      scanf(" %c", &more);
65
66      if(tolower(more) == 'r')
67          return false;
68
69      if(tolower(more == 's')){
70          record_search();
71          return false;
72      }
73      printf("\n%20cEnter the employee first name\n\n%32c", ' ', ' ');
74      getname(temp -> first_name);
75      printf("\n%20cEnter the employee last name\n\n%32c", ' ', ' ');
76      getname(temp -> last_name);
77      printf("\n%20cEnter the employee gross income\n\n%34c", ' ', ' ');
78      scanf("%d", &temp -> gross_income);
```

```
79        printf("\n%20cEnter the employee martial status\n\n%33c", ' ', ' ');
80        scanf("%s", &temp -> martial_status);
81        printf("\n\n\n");
82        return true;
83
84    }
85
86  void getname(char *name){
87
88        fflush(stdin);
89        fgets(name, NAME_MAX, stdin);
90        int len = strlen(name);
91        if(name[len - 1] == '\n')
92            name[len - 1] = '\0';
93    }
94
95  void show_person_data(void){
96
97        Employee member;
98        int sum = 0 , counter = 0;
99        float avg;
100       if(!(global.pfile = fopen(global.filename, "r"))){
101           perror("can not open file for reading");
102           exit(1);
103       }
104
105       printf("\n\n%30cRetrieving data\n\n", ' ');
106
107       while(fread(&member, sizeof member, 1, global.pfile)){
108               printf("\n%25cEmployee: %s %s\n%25cgross income: %d\n%25cmartial status: %s\n\n",
109                       ' ',member.first_name, member.last_name,' ',member.gross_income,' ',
110                        member.martial_status);
111               sum += member.gross_income;
112               counter++;
113       }
114       avg = sum / counter;
115       printf("%25cgross income average = %.2f\n\n",' ', avg);
116
117       fclose(global.pfile);
118       }
119
120 void record_search(void){
121
122       Employee member;
123       char name[NAME_MAX];
124       int len_str;
125
126       printf("%18cEnter the first name of the employee\n%20c",' ', ' ');
127       fflush(stdout);
128       scanf("%s", &name);
```

```
129
130         fclose(global.pfile);
131
132         if(!(global.pfile = fopen(global.filename, "r"))){
133             perror("can not open file for reading");
134             exit(1);
135         }
136
137         len_str = strlen(name);
138
139           if(name[len_str - 1] == '\n')
140             name[len_str - 1] = '\0';
141
142             while(fread(&member, sizeof member, 1, global.pfile)){
143                 if(strcmp(name, member.first_name) == 0)
144                     printf("\n%25cEmployee: %s %s\n%25cgross income: %d\n"
145                             "%25cmartial status: %s\n\n",' ',member.first_name,
146                              member.last_name,' ',member.gross_income,' ',
147                              member.martial_status);
148             }
149 }
```

شرح البرنامج

```
struct{
    char *filename;
    FILE *pfile;
}global = {"D:\\C\\C files\\mytext5.bin", NULL};
```

هنا تم تعريف هيكل و لم يتم إعطاؤه اسم معين و هذا مسموح به و لكن يجب تعريف أى متغير منه فى نفس الجملة كما هو فى هذه الحالة، و لا يمكن تعريفها بعد ذلك فى أى جملة أخرى، لأن الهيكل ليس له إم فكيف ستستخدمه فى إنشاء متغير جديد !

هذا الهيكل يحتوى على متغيرين و هما مؤشرين أحدهما يشير على الملف فى الهارد ديسك و أحدهما يشير على الملف فى الذاكرة ، و تم تعريف متغير منه و إعطاؤه تلك القيمة الإبتدائية ، كان بإمكاننا أن نقوم بهذا بالطريقة الإعتيادية دون الحاجة لهيكل، و لكن يتم إستخدام ذلك لتمييز هذه المتغيرات عما سواها بإستدعائها بكلمة global كما سنرى .

```c
typedef struct employee{
    char first_name[NAME_MAX];
    char last_name[NAME_MAX];
    int gross_income;
    char martial_status[NAME_MAX];

}Employee;
```

هنا تم تعريف هيكل جديد اسمه employee و هذا الهيكل سيحتوى على بيانات الموظف .

```c
bool get_person(Employee *pemployee);           /* Input function */
void getname(char *name);                      /* Read a name */
void show_person_data(void);                  /* Output function */
void record_search(void);
```

هذه هى الـ prototype الخاصة بالدوال التى سنستخدمها فى البرنامج . دالة get_person() تقوم بإستقبال بيانات الموظف و يتم إعطاؤها مؤشر يشير إلى هيكل من النوع Employee ، و دالة getname() تقوم بإستقبال الإسم من المستخدم . أما الدالة الثالثة فلعرض جميع البيانات، و الدالة الرابعة لعمل بحث عن موظف معين و عرض بياناته.

```c
Employee member;

printf("\n%22c--------------------------\n", ' ');
printf("%22cWelcome to Database Program\n", ' ');
printf("%22c---------------------------\n\n", ' ');

if(!(global.pfile = fopen(global.filename, "wb"))){
    perror("can not open file to write");
    exit(1);
}
```

هنا تم البدء فى كتابة الدالة الرئيسية للبرنامج ، و تم تعريف فيها متغير من النوع Employee و اسمه member ، ثم طباعة شكل البرنامج ، ثم فتح الملف للإستعداد لإدخال أى بيانات فيه.

```
while(get_person(&member))
    fwrite(&member, sizeof member, 1, global.pfile);

    fclose(global.pfile);
    show_person_data();

    if(remove(global.filename)){
        perror("can not delete the file");
        exit(1);
    }

    return 0;
}
```

هذه الحلقة التكرارية تقوم بكتابة ما يتم إدخاله من بيانات إلى الملف ، و تنتهى هذه الحلقة عندما تعود دالة get_person() بـ false و هى تعود بتلك القيمة عند الانتهاء من إدخال البيانات، و يتم إعطاء دالة get_person() عنوان المتغير member ليقوم بحفظ البيانات المدخلة فيه.

هنا يعمل الهيكل كوسيط تخزين يقوم بتخزين البيانات المدخلة من المستخدم ثم يتم بعد ذلك حفظ هذه البيانات إلى الملف للإحتاظ بها بصفة دائمة، و قد تطرقنا قبل ذلك إلى كيفية عمل fwrite() ، و هنا نوضح إستخدامها عند استقبالها البيانات من هيكل و كتابة هذه البيانات إلى ملف.

المعامل الأول لـ fwrite() هو عنوان أول متغير فى الهيكل member و المتغيرات التى يحتويها أى هيكل توضع متتابعة فى الذاكرة. فهنا سيتم قراءة جميع المتغيرات الخاصة بالهيكل بصورة تتابعية بداية من المتغير الأول.

سنقوم بقراءة هيكل واحد فى كل مرة لذا استخدمنا 1 ، ثم ادخلنا مؤشر الملف عن طريق إستدعاؤه بإسم الهيكل الذى يحتويه global .

ثم يتم إغلاق الملف مباشرة بعد الإنتهاء من عملية الكتابة ، و طباعة جميع البيانات عن طريق دالة show_person_data() ، ثم فى النهاية يتم حذف الملف من الهارد ديسك – يمكنك عدم حذفه إذا أردت الإحفاظ بالبيانات المُدخلة على الهارد ديسك الخاص بك.

سنتناول الآن شرح عمل كل دالة على حدة، و هذا سيوضح لك عمل البرنامج أكثر.

دالة get_ person

```
bool get_person(Employee *temp){

    static char more = '\0';

    printf("%26cSELECT AN OPERATION\n\n", ' ');
    printf("%5c(A: adding an employer, R: retreiving data,"
           "S: search for record)\n\n%35c", ' ',' ');
    scanf(" %c", &more);

    if(tolower(more) == 'r')
        return false;

    if(tolower(more == 's')){
        record_search();
        return false;
    }
```

الدالة get_person الخاصة بإستقبال البيانات من المستخدم ، كما نرى أن الدالة تختبر العملية المراد تنفيذها ، إذا كانت العملية هى R المختصة بعرض جميع البيانات التى أُدخلت مُسبقاً فالدالة تعود بـ false ، لأن فى هذه الحالة لن يتم إدخال بيانات فلا حاجة لتنفيذ باقى أوامر الدالة . و أما إذا كان إختيار المستخدم S ، فإن الدالة تستدعى دالة البحث عن موظف معين – سنتناول شرح هذه الدالة لاحقاً – ثم تعود كذلك بـ false .

إذا وقع الإختيار على A ، فستقوم الدالة بإستكمال تنفيذ أوامرها بإستقبال البيانات من المستخدم ، و يتم إستقبال الأسماء سواء الإسم الأول للموظف أو لقبه عن طريق دالة getname – التى سنتناول شرحها لاحقاً- أما باقى البيانات فيتم إستقبالها عن طريق scanf و توضع فى المتغيرات الخاصة بها فى الهيكل employee الذى يستخدم المؤشر temp فى الإشارة إليه ، ثم تعود الدلة فى النهاية بـ true .

```
printf("\n%20cEnter the employee first name\n\n%32c", ' ', ' ');
getname(temp -> first_name);
printf("\n%20cEnter the employee last name\n\n%32c", ' ', ' ');
getname(temp -> last_name);
printf("\n%20cEnter the employee gross income\n\n%34c", ' ', ' ');
scanf("%d", &temp -> gross_income);
printf("\n%20cEnter the employee martial status\n\n%33c", ' ', ' ');
scanf("%s", &temp -> martial_status);
printf("\n\n\n");
return true;
```

دالة getname

```c
void getname(char *name){

    fflush(stdin);
    fgets(name, NAME_MAX, stdin);
    int len = strlen(name);
    if(name[len - 1] == '\n')
        name[len - 1] = '\0';
}
```

دالة getname الخاصة بإستقبال أى بيانات حرفية . فى البداية نستخدم دالة fflush لعدم حفظ أى مسافات أدخلت قبل كتابة الإسم ، ثم نستخدم دالة fgets و هى إحدى الدوال التى تستخدم فى إستقبال البيانات ، و يكون المعامل الأول لها إسم المتغير الذى سيتم حفظ البيانات فيه ،و المعامل الثانى هو أقصى عدد من الحروف المدخلة ،و المعامل الثالث هو الجهة التى سيتم إستقبال البيانات منها ، و هنا سنستقبل البيانات من أداة الإدخال الإعتيادية و يرمز بها stdin و غالباً ما تكون هى لوحة المفاتيح . ثم يتم إختبار آخر حرف من الكلمة و إستبدالها إذا كانت n\ بـ 0\ لتكون البيانات مسجلة فى الملف بشكل سليم، تنتهى كل كلمة فيه بـ 0\ و ليس بـ n\ التى تستخدمها دالة fgets لإنهاء الكلمات المدخلة عن طريقها.

دالة show_person_data

```c
void show_person_data(void){

    Employee member;
    int sum = 0 , counter = 0;
    float avg;

 if(!(global.pfile = fopen(global.filename, "r"))){
     perror("can not open file for reading");
     exit(1);
 }

 printf("\n\n%30cRetrieving data\n\n", ' ');

 while(fread(&member, sizeof member, 1, global.pfile)){
        printf("\n%25cEmployee: %s %s\n%25cgross income: %d\n%25cmartial status: %s\n\n",
               ' ',member.first_name, member.last_name,' ',member.gross_income,' ',
                member.martial_status);
        sum += member.gross_income;
        counter++;
 }
```

```c
avg = sum / counter;
printf("%25cgross income average = %.2f\n\n",' ', avg);

fclose(global.pfile);
}
```

دالة show_person_data الخاصة بعرض جميع البيانات الموجودة للموظفين و كذلك متوسط الدخل . يتم تعريف متغير من الهيكل employee و هو member ، لإستخدامه كوسيط يتم حفظ فيه بيانات الموظف الواحد من الملف ثم عرضها للمستخدم ، ثم إستقبال بيانات موظف جديد فيه ثم عرضها و هكذا ، فهو يعمل كـ buffer .

نقوم بفتح الملف للقراءة ، ثم نقوم بالمرور على بيانات الموظفين بالملف و طباعة بيانات كل موظف على حدة عن طريق member التى تعمل كـ buffer كما وضحنا سابقاً ،و تستمر هذه العملية عند الإنتهاء من جميع بيانات الموظفين الموجودة فى الملف أى عندما تعود fread بـ **0** . و طريقة حساب متوسط الأجور لا يحتاج لإيضاح . ثم يتم إغلاق الملف.

دالة record_search

```c
void record_search(void){

    Employee member;
    char name[NAME_MAX];
    int len_str;

    printf("%18cEnter the first name of the employee\n%20c",' ', ' ');
    fflush(stdout);
    scanf("%s", &name);

        while(fread(&member, sizeof member, 1, global.pfile)){
            if(strcmp(name, member.first_name) == 0)
                printf("\n%25cEmployee: %s %s\n%25cgross income: %d\n"
                       "%25cmartial status: %s\n\n",' ',member.first_name,
                        member.last_name,' ',member.gross_income,' ',
                        member.martial_status);
        }
}
```

دالة record_search الخاصة بالبحث عن بيانات موظف معين عن طريق إسمه الأول . تقوم بطلب إدخال إسم الموظف ثم تحفظه فى المتغير name . ثم تقوم بفتح الملف للقراءة ، و مقارنة جميع أسماء الموظفين

بالإسم المراد البحث عنه ، و إذا تحقق الشرط يتم طباعة بيانات هذا الموظف .

هنا نكون قد إنتهينا من شرح البرنامج التطبيقى ، يُفضل أن يتم عمله بالإعتماد على نفسك ، و إدخال عليه تعديلات إذا أردت أن يقوم البرنامج بمهام أخرى . الآن مع التمارين.

التمارين

1) اكتب برنامجاً يقوم بإستقبال مجموعة من أسماء الطلاب ، و يقوم بحفظها فى ملف ، لا تقوم بحذف الملف بعد الإنتهاء .

2) اكتب برنامجاً يقوم بطباعة البيانات الموجودة فى الملف الذى تم إنشاؤه فى المثال السابق ، و لكن بصورة عكسية ، أى أن آخر اسم فى الملف يطبع أول اسم و هكذا .

3) اكتب برنامجاً يقوم بصناعة قاعدة بيانات لمجموعة مختلفة من موديلات السيارات بإستخدام الهيكل البيانى structure ، يقوم المستخدم بإدخال جميع البيانات ، و البرنامج يتيح للمستخدم طباعة كل البيانات التى تم إدخالها حتى الآن.

نهاية الكتاب

ما أصابني في هذا الكتاب من توفيق فمن الله عز و جل

و ما أصابني من زلل أو تقصير فمن نفسي أو الشيطان

اللهم اجعله عملاً خالصاً لوجهك الكريم يارب العالمين .